Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Facultad de Informática

TRABAJO FIN DE GRADO

Implementación del modelo RIM de HL7 v3 en orientación a objetos y su uso en procesos de interoperabilidad semántica.

Autor: Enrique Alonso Oset

Tutor: Raúl Alonso Calvo

POLITÉCNICA
"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MADRID, JUNIO DE 2014

Índice de contenidos

1 RESUMEN DEL TRABAJO REALIZADO

Este Trabajo de Fin de Grado ha sido realizado por Enrique Alonso Oset, alumno de Grado en Ingeniería Informática en la Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid bajo la supervisión del tutor del proyecto, Raúl Alonso Calvo. El trabajo se ha desarrollado durante el segundo semestre del curso académico 2013/2014.

El trabajo ha sido realizado dentro del marco de los proyectos EURECA (Enabling information re-Use by linking clinical REsearch and Care) e INTEGRATE (Integrative Cancer Research Through Innovative Biomedical Infrastructures), en los que colabora el Grupo de Informática Biomédica de la UPM junto a otras universidades e instituciones sanitarias europeas. En ambos proyectos se desarrollan servicios e infraestructuras con el objetivo principal de almacenar información clínica, procedente de fuentes diversas (como por ejemplo de historiales clínicos electrónicos de hospitales, de ensayos clínicos o artículos de investigación biomédica), de una forma común y fácilmente accesible y consultable para facilitar al máximo la investigación de estos ámbitos, de manera colaborativa entre instituciones. Esta es la idea principal de la interoperabilidad semántica en la que se concentran ambos proyectos, siendo clave para el correcto funcionamiento del software del que se componen. El intercambio de datos con un modelo de representación compartido, común y sin ambigüedades, en el que cada concepto, término o dato clínico tendrá una única forma de representación. Lo cual permite la inferencia de conocimiento, y encaja perfectamente en el contexto de la investigación médica.

En concreto, la herramienta a desarrollar en este trabajo también está orientada a la idea de maximizar la interoperabilidad semántica, pues se ocupa de la carga de información clínica con un formato estandarizado en un modelo común de almacenamiento de datos, implementado en bases de datos relacionales.

El trabajo ha sido desarrollado en el periodo comprendido entre el 3 de Febrero y el 6 de Junio de 2014. Se ha seguido un ciclo de vida en cascada para la organización del trabajo realizado en las tareas de las que se compone el proyecto, de modo que una fase no puede iniciarse sin que se haya terminado, revisado y aceptado la fase anterior. Exceptuando la tarea de documentación del trabajo (para la elaboración de esta memoria), que se ha desarrollado paralelamente a todas las demás.

A continuación se describen las tareas, y el tiempo dedicado en cada una de ellas:

- Formación: Adquisición de conocimientos, de tecnologías y contexto [35 horas].
- Especificación de requisitos: Funcionalidad requerida y restricciones de la herramienta [32 horas].
- Diseño: Decisiones, estructura y organización de la implementación [40 horas].

- Implementación del prototipo: Implementación de los módulos especificados en los objetivos [120 horas].
- Validación de la herramienta: Ejecución de pruebas del sistema y detección de errores [35 horas].
- Depuración de la herramienta: Implantación de mejoras y optimizaciones para arreglar defectos de la herramienta [30 horas].
- Elaboración de la memoria: Documentación del trabajo realizado [49 horas].

This Bachelor Project has been performed by Enrique Alonso Oset, student of a degree in Computer Science in the Informatics University (ETSIINF) of the UPM, Madrid, in collaboration with the project's tutor, Raúl Alonso Calvo. The project has been developed during the second semester of the 2013/2014 academic year.

This Project has been done inside EURECA and INTEGRATE European biomedical research projects, where the GIB (Biomedical Informatics Group) of the UPM works as a partner. Both projects aim is to develop platforms and services with the main goal of storing clinical information (e.g. information from hospital electronic health records (EHRs), clinical trials or research articles) in a common way and easy to access and query, in order to support medical research.

The whole software environment of these projects is based on the idea of semantic interoperability, which means the ability of computer systems to exchange data with unambiguous and shared meaning. This idea allows knowledge inference, which fits perfectly in medical research context.

The tool to develop in this project is also "semantic operability-oriented". Its purpose is to store standardized clinical information in a common data model, implemented in relational databases.

The project has been performed during the period between February 3rd and June 6th, of 2014. It has followed a "Waterfall model" of software development, in which progress is seen as flowing steadily downwards through its phases. Each phase starts when its previous phase has been completed and reviewed. The task of documenting the project's work is an exception; it has been performed in a parallel way to the rest of the tasks.

A short description of the project's tasks and it's time duration is provided next:

- Initiation: Knowledge acquirement about project's technologies, context [35 h].
- Software requirements specification: Functionality and restrictions [32 h].
- Design: Implementation phase's decisions, structures and organization [40 h].
- Construction: Implementing the required modules [120 h].
- Testing: Executing the complete tool and detecting flaws [35 h].
- Maintenance: Modification of the software product after it's testing to correct faults [30 h].
- Documenting: Register the entire project's info in an academic document [49 h].

2 INTRODUCCIÓN

El objetivo principal de este Trabajo de Fin de Grado es emplear programación orientada a objetos (Java) para implementar el modelo RIM (Reference Information Model) del estándar HL7 v3. De modo que se puedan aplicar las ventajas y características de la programación orientada a objetos para trabajar con bases de datos del RIM de HL7 v3.

HL7 (Health Level 7) es una organización internacional de desarrollo de estándares y herramientas de interoperabilidad para facilitar el intercambio electrónico de conocimiento e información clínica. En concreto, el estándar de HL7 versión 3 centra el intercambio de información en el RIM, una especificación estructurada de la información dentro del escenario de la salud, que ofrece la posibilidad de representar todo tipo de eventos relacionados con el ámbito clínico.

El presente Trabajo de Fin de Grado se enmarca dentro de los proyectos INTEGRATE y EURECA, en los que colabora el Grupo de Informática Biomédica de la UPM. En dichos proyectos el RIM de HL7 v3 se ha usado como base para desarrollar un componente denominado Common Data Model (CDM), con un modelo relacional para bases de datos en el que almacenar de forma conjunta y homogénea información biomédica de diversas fuentes e instituciones con distintas representaciones. El modelo se actualiza (siguiendo especificaciones del RIM) en función de las necesidades de representación para nuevos datos que se van obteniendo. Es un componente clave para proporcionar interoperabilidad semántica entre sistemas informáticos de investigación biomédica.

En este trabajo se implementarán las entidades del CDM en clases de Java, de forma que se puedan emplear dichas clases para realizar, mediante lenguaje Java, funciones como procesos ETL de extracción, transformación y carga de datos a partir de mensajes HL7 CDA (documento XML con una estructura estandarizada para el intercambio de datos clínicos) u otras fuentes (bases de datos en otros modelos, hojas de cálculo, ficheros XML que no sigan el estándar CDA, etc…). Para ello se necesita hacer uso de conexiones JDBC con bases de datos basadas en el RIM, y de alguna herramienta o librería para Java como Hibernate, que permite el *mapping* entre objetos Java y entidades de bases de datos, mediante archivos XML que almacenen la información de mapeo, o mediante anotaciones en las clases que representan a las entidades. Hibernate permite guardar o actualizar de forma persistente los objetos de datos en la BBDD, o consultar información de objetos ya almacenados.

El proyecto cuenta con los siguientes objetivos:

- Elaborar una especificación de requisitos de la herramienta que satisfaga las necesidades planteadas.
- Implementación de las clases del CDM en Java.

- Implementación de un módulo para la inserción automática de mensajes HL7 CDA en una base de datos biomédica usando los objetos RIM creados.
- Validación de la herramienta mediante la ejecución automática de una batería de consultas.

Esta memoria presenta una sección de especificación de requisitos software que ha de cumplir la herramienta a desarrollar en este Trabajo de Fin de Grado para satisfacer todas las necesidades planteadas.

Contiene también una sección de diseño con toda la información relativa a la fase de diseño del proyecto. En ella se expone el plan a seguir en la fase de implementación, realizando una descripción de los detalles internos de cada componente que tendrá el sistema, así como su organización y las interfaces entre componentes. Se tendrán en cuenta las restricciones impuestas en la fase de especificación de requisitos.

En la sección de desarrollo se describen las discrepancias con estas dos fases previas y todos los detalles relativos a la implementación de cada componente de la herramienta.

La sección de pruebas del sistema documenta la implementación del módulo de pruebas, independiente de la herramienta.

En las fases de conclusiones y futuras líneas de actuación, se exponen resultados y detalles relevantes surgidos durante el desarrollo del trabajo, así como los próximos pasos a realizar relacionados con la herramienta implementada.

Y finalmente se detallan las referencias consultadas para la formación y la resolución de problemas para realizar este trabajo. El formato de estas referencias sigue las recomendaciones de la IEEE.

Todo el trabajo documentado en esta memoria ha sido realizado por el alumno, Enrique Alonso Oset. En los casos en lo que algo se haya desarrollado de manera automática, apoyándose en algún servicio determinado, o utilizando un trabajo ya realizado por una entidad ajena se ha especificado claramente la procedencia o autoría de dicho trabajo.

3 TRABAJOS PREVIOS Y CONTEXTO

La herramienta a desarrollar quedará integrada dentro del entorno software de los proyectos EURECA e INTEGRATE, donde se han realizado numerosos trabajos y aplicaciones orientadas a la interoperabilidad semántica entre modelos de datos para la gestión de ensayos clínicos. En la sección 4.4.1 se describe la perspectiva de la herramienta a desarrollar, su función y cómo encaja dentro del entorno, así como la descripción de los elementos con los que interaccionará.

Actualmente, la labor de cargar mensajes XML con el formato establecido por el estándar CDA de HL7 v3 en el modelo común de datos de los proyectos EURECA e INTEGRATE, ya la realiza una herramienta desarrollada en el marco de estos proyectos. Esta herramienta está implementada con Mirth Connect [49] [50], un sistema multiplataforma que actúa como interfaz de HL7 para el intercambio de información clínica estandarizada, permitiendo el envío bidireccional de mensajes HL7 entre sistemas y aplicaciones a través de múltiples protocolos o sistemas de transporte. Mirth Connect presenta una arquitectura basada en canales, y en concreto, la implementación realizada consiste en un canal con una conexión configurable a una base de datos del CDM en MySQL en la que carga los mensajes que reciba como entrada, verificándolos previamente.

Mirth Connect también se emplea para la generación de los mensajes HL7 a partir de fuentes de datos originales, lo que demuestra su carácter bidimensional.

Se ha querido desarrollar una herramienta que realice esta misma labor, pero con la implementación del modelo común de datos en objetos Java, para aprovechar las numerosas posibilidades que la programación orientada a objetos puede ofrecer. A la hora de implementar la herramienta se ha tratado de emular todas las funcionalidades que ofrece Mirth Connect, como por ejemplo, realizar un análisis previo de los mensajes, viendo que cumplan con el estándar de HL7 para la construcción de mensajes CDA, para lo que se ha implementado un esquema XSD con todas las estructuras posibles de los mensajes, y añadir otras funcionalidades y ventajas como hacer una herramienta que sólo dependa de una plataforma (MySQL). En la sección de desarrollo se dan ejemplos de las comodidades que han ofrecido a la hora de realizar operaciones con bases de datos.

También puede ofrecer futuros usos que pueden ser útiles en el contexto de los proyectos en los que se enmarca este TFG. Como por ejemplo el uso de los objetos como formato de intercambio de información clínica, permitiendo abstraerse del formato XML, y pudiéndolos cargar directamente en el CDM con el uso de Hibernate. Este distinto formato, basado en clases, para la transferencia de información clínica también está contemplado en el estándar de HL7 v3, y se denomina Refined Message Information Models (R-MIMs) [21].

Otro posibilidad futura puede ser el uso de la representación de bases de datos del CDM en objetos para realizar consultas de forma más sencilla con lenguaje Java o para realizar arreglos en los datos (por ejemplo en anotaciones incorrectas escogidas para representar un cierto concepto de un ensayo clínico) con actualizaciones en la base de datos sencillas de implementar mediante Java.

4 ESPECIFICACIÓN DE REQUISITOS

4.1 Introducción

En este capítulo se detalla la Especificación de Requisitos Software (ERS) del sistema a desarrollar en el Trabajo de Fin de Grado.

El propósito de esta especificación es reflejar todas las restricciones bajo las que el sistema propuesto debe operar, y las funcionalidades que debe ofrecer para satisfacer los objetivos marcados para este proyecto, qué datos empleará y qué resultados deberá producir. También se comentarán el fin de la creación de esta herramienta y los beneficios que aportará, su interacción con los usuarios finales describiendo sus interfaces, y sus reacciones frente a estímulos exteriores. Una vez finalizada y revisada, esta especificación servirá de base en las fases posteriores de diseño e implementación del proyecto.

En esta etapa se acordará todo lo que se requiere del sistema y prevalecerá para las siguientes etapas, no pudiéndose requerir nuevos resultados a mitad del proceso de elaboración del software.

En ningún caso esta sección describirá decisiones o planes de implementación ni detalles internos de sus componentes, si no que se centrará en las funciones externas visibles y atributos del sistema. La descripción de cada componente y del plan de implementación se realizará en la fase del diseño del proyecto.

La especificación constará de una sección de ámbito y alcance del proyecto en la que se realiza una descripción general de en qué consiste el sistema a implementar y el contexto en el que se engloban sus requisitos, las posibilidades que ofrecerá y los objetivos y beneficios que se pretenden alcanzar con su uso. También contendrá una sección explicativa con un glosario de términos relevantes en el contexto del sistema. La última sección, de descripción global del sistema consta de una subsección explicativa de la perspectiva del producto, en la que se explica la relación e interacción del sistema a implementar con otros ya implementados dentro de los proyectos EURECA e INTEGRATE en los que se engloba el proyecto, y una subsección final en la que se explican detalladamente todas las *features* o prestaciones que establecen el comportamiento del sistema, cada una dividida en varios requisitos funcionales, que definen una condición o capacidad sobre el contenido, forma o funcionalidad del sistema para satisfacer los objetivos impuestos.

Esta especificación está realizada siguiendo las directrices definidas en el estándar "IEEE Recommended Practices for Software Requirements Specification, IEEE Std. 830-1998" [13][14]. Este estándar describe las estructuras posibles, contenido deseable, y calidades de una especificación de requisitos del software.

El documento va dirigido tanto al tutor del proyecto como a los usuarios finales de la herramienta a desarrollar, que son miembros de los proyectos EURECA e INTEGRATE, por lo que contiene alguna discrepancia con el estándar por tratarse de una especificación de requisitos más realista que perfecta, como puede ser el caso de incluir algún detalle técnico más relativo a la fase de diseño.

4.2 Ámbito y alcance

En este proyecto se pretende crear una herramienta que pueda insertar de manera automática mensajes XML que sigan el estándar de información clínica HL7 CDA en bases de datos del modelo RIM de HL7 v3 con conexiones fácilmente configurables. Los usuarios podrán registrar información clínica estandarizada en mensajes HL7 en una base de datos de un modelo común de datos. La herramienta reconocerá multitud de estructuras posibles de mensajes del estándar CDA pero no reconocerá toda la información que contiene el modelo RIM de HL7 si no únicamente la que se utiliza en la versión adaptada del RIM desarrollada y empleada en los proyectos EURECA e INTEGRATE en los que se enmarca este proyecto. Dicho modelo se denomina CDM (Common Data Model).

Internamente, la aplicación a través de representación de las entidades del modelo con clases Java creará objetos con los datos de los mensajes y los cargará en la base de datos. Esto a pesar de ser una decisión de implementación (más conveniente de explicar en la fase de diseño) también supone un requisito impuesto, ya que posibilita la realización de una herramienta más eficiente, que permite el uso de características propias de la orientación a objetos (como herencia y polimorfismo) y ofrece una mayor posibilidad de reutilización y mejor soporte de cambios en el código fuente para añadir nueva funcionalidad por ejemplo si se producen cambios en el modelo.

Los principales objetivos son representar fielmente, con una correspondencia adecuada de tipos, las entidades del modelo en clases java, y mediante el uso de esas clases, procesar mensajes HL7 para cargar su contenido en el modelo de manera eficiente y sencilla, e informando al usuario durante la ejecución del progreso de la carga de datos mostrando estadísticas de los mensajes procesados correctamente, y las secciones con errores o ignoradas por tener información desconocida. La herramienta realizará una validación previa de dichos mensajes e informará al usuario de si están construidos correctamente y a continuación realizará la carga de datos.

La herramienta pasará a denominarse ETL (*extract, transform and load*), porque realiza un proceso de extracción de información de los mensajes reconociendo sus estructuras, transformación (creación de objetos java almacenando en sus atributos dicha información extraída), y carga en una base de datos (persistencia de dichos objetos).

4.3 Glosario de términos

- **ETL:** Extract, transform and load. Proceso de obtención de datos de una fuente, procesado y reformateado de los mismos para posteriormente almacenarlos en una base de datos, en este caso de datos clínicos procedentes de mensajes estandarizados. Nomenclatura elegida para referirse a la herramienta a desarrollar en el proyecto.

- **HL7:** Health Level 7, es una organización internacional de desarrollo de estándares y herramientas de interoperabilidad para facilitar el intercambio electrónico de conocimiento e información clínica.

- **HL7 CDA:** Clinical Document Architecture. Estándar creado por la organización HL7 para la arquitectura clínica de documentos. Es un estándar basado en XML para el marcaje de documentos que especifica la estructura y semántica de documentos clínicos para el propósito de facilitar su intercambio en un entorno de interoperabilidad.

- **Mensaje HL7 CDA:** Unidad de trabajo de la herramienta, fichero XML que contiene información clínica de un mismo paciente estructurada según el estándar CDA.

- **HL7 RIM:** Modelo de Referencia de Información del estándar HL7 v3. Especificación estructurada de la información dentro del escenario de la salud. Contiene un modelo de clases que permite contextualizar cualquier evento del ámbito de la salud. A partir del RIM, se construyen las especificaciones de mensajes específicos para diferentes dominios del escenario de salud.

- **CDM:** Modelo común de datos. Modelo entidad-relación de bases de datos desarrollado basándose en la especificación del Reference Information Model (RIM) del estándar HL7 v3. Es el modelo empleado para almacenar los datos procedentes de todas las organizaciones colaboradoras de los proyectos EURECA e INTEGRATE. La carga de datos se realizará en una base de datos que siga este modelo.

- **CD:** Core Dataset. Conjunto de terminologías de las que se compone el vocabulario estándar aceptado para toda la información que se incluye en el CDM. Estas terminologías son SNOMED-CT, HGNC y LOINC.

- **IHE:** Integrating the Healthcare Enterprise. Iniciativa de empresas y profesionales de la sanidad cuya finalidad es mejorar la comunicación entre los distintos sistemas de información sanitarios. IHE no desarrolla nuevos estándares, sino que promueve el uso coordinado de estándares ya existentes, como DICOM, XML y HL7 para resolver necesidades específicas de los clínicos y mejorar la calidad de la atención a los pacientes.

- **Validación:** Comprobación de que un documento en lenguaje XML está bien formado y se ajusta a una estructura definida. En este caso, los documentos analizados deben seguir tanto las reglas dictadas para XML como las definidas por el estándar CDA.

- **Mapping:** Proceso de creación de elementos que contengan la traducción entre dos modelos de datos distintos. En el caso que nos ocupa, habrá que crear ficheros XML que permitan la traducción entre tablas de SQL y objetos java, con una correspondencia de tipos, almacenando la procedencia de los campos, etc…

- **Cabecera del mensaje:** El resultado de la instancia de CDA estará compuesto por una cabecera codificada con XML, y el cuerpo. La cabecera contiene datos identificativos del documento (código identificador único en la etiqueta typeID, tipo de documento, versión), código identificador y datos demográficos del paciente (en la etiqueta recordTarget) como fecha de nacimiento, género, nombre y apellidos. También puede incluir información referente a otros participantes como médicos u organizaciones.

- **Cuerpo del mensaje:** El Cuerpo contiene información narrativa sobre el sujeto del documento, normalmente un paciente (es donde está la información clínica). Cada documento CDA tiene exactamente un cuerpo, asociado con la clase ClinicalDocument a través de la relación component. Un cuerpo de un CDA puede ser representado a través de un cuerpo estructurado o uno no estructurado en XML. Un contenido XML estructurado siempre está insertado dentro de un elemento structuredBody. Un cuerpo estructurado está compuesto por uno o más

elementos component, que pueden estar compuestos a su vez por ninguna o varias secciones (Section), compuestas a su vez de entradas (entry). La herramienta reconocerá cuerpos estructurados únicamente.

- **Entry:** En un cuerpo estructurado, la información clínica se estructura y define a través de clases tipo entry, que suponen cada uno de los actos en los que se ve involucrado un paciente. Pueden tratarse de observaciones (como diagnósticos o mediciones), procedimientos o administraciones de sustancias. El estándar CDA define más posibles entradas pero no podrán ser procesadas por la herramienta ya que se trata de información no incluida en el CDM. En el interior de cada entrada se almacena un campo con el tipo de acto correspondiente (etiquetas observation, procedure o substanceadministration) con un valor en el atributo classCode que identifique el tipo de acto (los posibles valores son 'OBS', 'SBADM' y 'PROC').

- **Códigos:** Secuencia generalmente numérica (dependiendo de la terminología), que identifica un concepto en una terminología. Se almacena como atributo de una etiqueta y suele ir acompañado de otros atributos como codeSystem y codeSystemName en los que se almacena el nombre y el código identificativo de la terminología de la que se ha extraído el código del concepto, y un atributo displayName en el que se almacena el título del concepto utilizado. Todos los conceptos almacenados en el CDM deben formar parte de las terminologías SNOMED-CT, LOINC o HGNC. A continuación se muestra un ejemplo:

```
<code code="395557000" codeSystem="2.16.840.1.113883.6.96"
codeSystemName="SNOMED CT" displayName="Tumor finding"/>
```

- **Effective Time:** Tiempo relevante en el ocurre un acto. Es decir, por ejemplo, en el caso de una observación es el tiempo en el que la observación es (ha sido) efectiva para el paciente

- **OIDs:** Identificadores ISO admitidos por HL7. Están divididos en dos componentes, root (identificador global cuya raíz está asignada por la ISO, o ha sido obtenida desde HL7) y extensión (valor asignado por la organización o aplicación donde se crea el documento), cuya concatenación supone una cadena única que identifica el documento.

- **Usuarios del sistema:** investigadores de los proyectos EURECA e INTEGRATE que cargan en sus bases de datos del CDM los datos clínicos que reciben de instituciones colaboradoras en mensajes estandarizados HL7 CDA.

- **SNOMED-CT:** Ontología de términos clínicos que asocia un código numérico a cada concepto y está estructurada en ramas de categorías diversas.

- **LOINC:** Terminología empleada para diagnósticos, mediciones, observaciones de laboratorios, etc…

- **HGNC:** Terminología empleada para información genética.

- **JDBC:** *Java Database Connectivity*. API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

4.4 Descripción global del sistema

4.4.1 Perspectiva

La herramienta a desarrollar no será un sistema independiente y formará parte del entorno software de los proyectos EURECA e INTEGRATE, interactuando con otros servicios desarrollados en dichos proyectos.

En primer lugar deberá integrarse con sistemas de recogida y organización de datos procedentes de historiales clínicos de hospitales o de ensayos clínicos. Posteriormente también requiere que se empleen herramientas de mapeo o anotación de los datos en unas terminologías determinadas, y generación automática de mensajes estructurados HL7 CDA con dichos datos.

Una vez generados los mensajes se le entregarán como entrada a la herramienta, que extraerá su información y la cargará en el *Common Data Model*.

La herramienta coexiste con los dos elementos fundamentales de la capa semántica desarrollada en ambos proyectos, el *CDM* y el *Core Dataset*, representando al primero con clases java, y empleando conceptos del segundo para toda la información que

procesa. Aunque la herramienta puede aceptar entradas en otras terminologías para las que posteriormente se buscará una traducción.

Después de la carga se ejecutará sobre la base de datos resultante un proceso de normalización, con un sistema denominado *Normalization Pipeline*, cuyo objetivo es crear una nueva base de datos transformando los datos contenidos en la anterior de manera que únicamente tenga conceptos correctos de las terminologías *SNOMED-CT, LOINC y HGNC*. Este proceso puede provocar la creación de nuevos conceptos mediante la división de algunos conceptos de la base de datos inicial para que su información sea más completa (en función de sus categorías y la información que representan), la corrección de conceptos mal asignados al CDM (pudiendo cambiar las tablas en los que éstos se almacenan), la creación de las relaciones que correspondan a los nuevos conceptos creados, o la traducción de los conceptos anotados en terminologías que no sean las que componen el *Core Dataset*.

Finalizado este proceso se establece un punto de acceso para realizar consultas semánticas en *SPARQL* y recuperar la información almacenada.

Una vez esté finalizada la implementación de la herramienta, se podrá incluir dentro del *Data Push Service*, creando un cliente web para ejecutar la herramienta de manera remota.

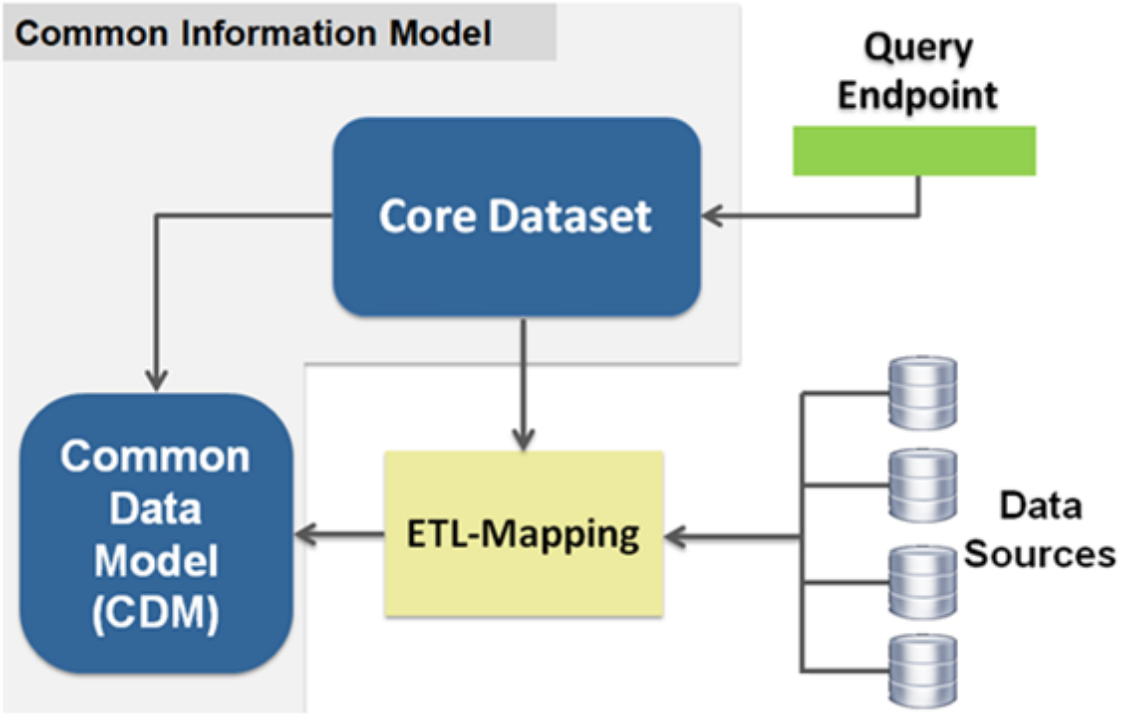


Figura 1. Diagrama de organización de los elementos con los que coexiste la herramienta a desarrollar. Modelo Común de Información.

4.4.2 Requisitos funcionales

A continuación se presenta la lista de las prestaciones (*features*) que el sistema deberá satisfacer. Para cada característica o *feature* se detallan uno o más requisitos del software, adecuadamente numerado. Estos requisitos ayudan a establecer el comportamiento del sistema.

4.4.2.1 Feature #1: Representación del CDM con objetos Java

1.1 La herramienta a desarrollar estará compuesta por un módulo con la representación completa de las entidades del CDM en objetos Java.

1.2 Se debe prestar atención a la asignación de tipos de bases de datos relacionales a tipos de programación orientada a objetos en los atributos de los objetos resultantes. Para cada entidad en el modelo de origen se creará una clase Java con todos sus atributos y métodos para operar con ellos y un fichero XML que almacene la tabla y el nombre de la base de datos concreta de procedencia, y el *mapping*, es decir, la asignación de tipos. Como alternativa se podrá tener un único fichero Java para cada entidad, que contenga anotaciones XML con las traducciones empleadas.

1.3 El módulo que contiene la implementación en objetos de las entidades del CDM debe contener una conexión JDBC a una base de datos que siga este modelo. Se deberá mantener un fichero XML con información relativa a las propiedades de dicha conexión, como los datos de autenticación, el sistema gestor de bases de datos utilizado y por tanto el *driver* de Java empleado (elemento que permite a la aplicación Java interactuar con la base de datos), la URL de la conexión, el "dialecto" de SQL o lenguaje que emplea el sistema de la base de datos, y más opciones de configuración de la misma.

1.4 Se requiere el uso de alguna implementación de mapeo objeto-relacional para Java (como la librería Hibernate) que permita persistir los objetos creados en una BBDD basada en el modelo del CDM.

4.4.2.2 Feature #2: Preprocesado de mensajes (análisis y validación)

2.1 La herramienta indicará al cliente que especifique la ruta del directorio en el que se encuentran los mensajes a cargar en la base de datos. El cliente podrá seleccionar dicho directorio mediante el explorador de carpetas. Si selecciona un fichero en lugar de un directorio se mostrará un mensaje de error, pero se le permitirá al usuario volver a

seleccionar el directorio. Si el directorio contiene ficheros que no tengan extensión .xml la herramienta los ignorará.

2.2 La herramienta realizará una validación previa de los mensajes antes de extraer sus datos. Se comprobará que los mensajes sean documentos XML bien formados y que se ajusten a la estructura definida por el estándar HL7 CDA, es decir, que contengan los elementos mínimos necesarios en un mensaje HL7 CDA.

2.3 En primer lugar la herramienta parseará los mensajes uno a uno comprobando que estén bien formados, que contengan una primera línea de declaración XML, que exista un único elemento raíz y que la sintaxis de todos sus elementos sea la correcta.

2.4 En segundo lugar se realizará una validación de los mensajes, en función de las directrices del estándar, es decir, que contenga los elementos mínimos para el cuerpo y la cabecera del mensaje. A continuación se muestra la estructura de una cabecera para un mensaje, destacando los elementos mínimos (en negro para las etiquetas y rojo para los valores), y los opcionales (en azul):

```
<ClinicalDocument xmlns="urn:hl7-org:v3" xmlns:mif="urn:hl7-org:v3/mif"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:hl7-org:v3
CDA.xsd">
<typeId root="2.16.840.1.113883.1.3" extension="POCD_HD000040"/>
<id root="xxxx" extension="xxxx"/>
<code code="xxxx" codeSystem="xxxx" codeSystemName="xxxx" displayName="xxxx"/>
<effectiveTime value="xxxx"/>
<confidentialityCode code="x" codeSystem="xxxx"/>
<recordTarget>
  <patientRole>
    <id extension="xxxx" root="xxxx"/>
    <patient>
      <name>
        <given>xxxx</given>
        <family>xxxx</family>
        <family>xxxx</family>
      </name>
      <administrativeGenderCode code="x" codeSystem="xxxx"/>
      <birthTime value="xxxx"/>
    </patient>
    <providerOrganization>
      <id root="xxxx"/>
    </providerOrganization>
  </patientRole>
</recordTarget>
```

Figura 2. Estructura de la cabecera de un mensaje HL7 CDA.

La herramienta reconocerá cuerpos estructurados únicamente. Por tanto, la etiqueta *nonXMLBody* provocará un error y no se analizará el cuerpo del mensaje. A continuación se muestra la estructura básica del cuerpo de un mensaje, indicando sus elementos mínimos necesarios (dentro de las secciones habrá *entries*):

```
<component>
 <structuredBody>
        <component>
                <section>
                <code code="xxxx" codeSystem="xxxx"
                codeSystemName="xxxx" displayName="xxxx"/>
                <text>
                xxxx<content styleCode="xxxx">xxxx</content>xxxx
                </text>
        </section>
        </component>
    </structuredBody>
</component>
<!—Comentario: después de un cuerpo (estructurado o no), hay que cerrar el ClinicalDocument-->
</ClinicalDocument>
```

Figura 3. Estructura del cuerpo de un mensaje HL7 CDA.

2.5 La herramienta sólo aceptará el formato de fechas especificadas por el estándar, es decir, que los valores de todas las etiquetas effectiveTime deberán tener el siguiente formato: YYYYMMDD, sin ningún delimitador.

4.4.2.3 Feature #3: Inserción de los datos extraídos de los mensajes

3.1 Posteriormente, una vez analizados los mensajes, se utilizará la implementación del CDM en objetos para almacenar los datos biomédicos contenidos en ellos. Se crearan objetos para las entidades correspondientes del CDM almacenando en sus atributos los valores extraídos de los mensajes. Y finalmente se guardaran dichos objetos de forma persistente en una base de datos.

3.2 En primer lugar, se analizará la cabecera del mensaje. La información del documento se almacenará en un acto con valor “FILE” para su atributo *classCode*, dicho acto almacenará el identificador, la fecha y el título del documento. La información del paciente se almacenará en las tablas Entity, LivingSubject y Person (un objeto para cada tabla, todos con el mismo identificador). Almacenando su fecha de nacimiento, su identificador, y su género (tras traducirlo del vocabulario HL7 Administrativa Sex a SNOMED-CT), y se registrará como código de la entidad generada el concepto de sujeto viviente de la terminología SNOMED-CT.

3.3A continuación se procesará el cuerpo del mensaje, creando una participación con la entidad del paciente generada anteriormente para cada acto que le corresponda. Se generará un acto por cada *Entry* detectada en el cuerpo. Se generará un identificador aleatorio para cada acto y se almacenará toda la información en los atributos

correspondientes (código, título, fecha, etc…). En el caso de la fecha, se almacenará por defecto en el campo effectiveTimeStart.

3.4 Los actos pueden ser de 3 clases: observaciones, procedimientos o administraciones de sustancia (no se aceptarán otras). Se generará también un objeto para dichas tablas, con el mismo identificador que el generado para el acto. En el caso de ser una administración de sustancia, la entrada puede incluir información de la sustancia concreta, para la cual se generará un objeto de la clase *Entity* y una participación con el acto de administración de sustancia. También se almacenará el código de la sustancia en el campo de *codeOrig* del acto.

3.5 Dentro de las entradas también puede haber información que se deberá almacenar en tablas aparte, como por ejemplo *ActTargetSiteCode, ActObservationValues, ActObservationInterpretationCode o ActMethodCode,* que se almacenará en objetos que representen a dichas tablas, pero siempre utilizando el mismo identificador que el generado para el acto. La herramienta reconocerá todas las posibles combinaciones de información que pueda contener una entrada (también podría contener otros actos relacionados, que se crearían, junto a un objeto de *ActRelationship* que almacene la relación entre ambos).

3.6 Todos los actos generados se relacionarán con el acto que representa al documento (acto con classCode=”FILE”).

3.7 Si cualquier campo opcional tiene valor vacío se almacenará valor “*null*” en el atributo correspondiente.

3.8 La herramienta aceptara conceptos de vocabularios que no formen parte del *CoreDataset*. Generará un aviso, pero no un error.

3.9 Una vez cargada la información de todos los mensajes, se generará un acto que represente al estudio del que se han extraído los datos, con valor “CLNTRL” (*Clinical Trial*) en el atributo *classCode*.

4.4.2.4 Feature #4: Estadísticas e información mostrada al usuario

4.1 En primer lugar, al iniciar la ejecución se mostrará un mensaje indicando al usuario que se autentifique, respondiendo si los parámetros insertados son correctos o volviendo a solicitarle que los inserte si no son correctos.

4.2 A continuación se le permitirá seleccionar la ruta del directorio que contenga los mensajes a procesar, si no es correcta (no existe, no es un directorio), se mostrará un mensaje informando al usuario del error, y se le permitirá volver a seleccionar el directorio.

4.3 Se mostrará información del progreso de la validación de los mensajes, indicando las líneas que provoquen error y si no se produce, un mensaje en el que se especifique el nombre del fichero indicando que se ha procesado correctamente. Una vez finalizada la validación mostrará el número de mensajes validados correctamente y los que han provocado error.

4.4 Antes de comenzar la inserción se mostrará el nombre de la base de datos en la que se va a insertar, y un aviso si no está vacía, preguntando al usuario si desea continuar con la operación.

4.5 Durante la inserción, si el mensaje contiene entradas con información que no esté modelada en el CDM se ignorarán y se mostrará un aviso. La herramienta aceptara conceptos de vocabularios que no formen parte del Core Dataset, generará un aviso si esto ocurre.

4.6 Se irá informando al usuario del progreso de la carga de cada mensaje, y al finalizar se mostrará un mensaje en el que se especifique el nombre del fichero indicando que se ha procesado correctamente. Al finalizar la carga se mostrará el número de mensajes cargados y el número de mensajes con avisos.

4.7 Una vez finalizadas todas las fases de la ejecución se informará del tiempo requerido y se le dará al usuario la opción de guardar un fichero de log con las estadísticas de la ejecución.

4.4.2.5 Feature #5: Seguridad

5.1 Se le pedirá al usuario introducir el nombre de usuario y contraseña de acceso a la base de datos y se compararán los valores introducidos con los parámetros de la conexión. Si no coinciden, se mostrará un mensaje de error y finalizará la ejecución.

5.2 Antes de cargar datos se analizará si la base de datos está vacía y se mostrará un aviso al usuario si no lo está, preguntándole si desea continuar.

4.4.2.6 Feature #6: Interfaz gráfica [Opcional]

6.1 Como requisito opcional, una vez implementada toda la funcionalidad requerida, se desarrollará una interfaz gráfica para la herramienta. Será tiempo adicional al incluido en la planificación, no sustituirá nada del tiempo planificado para otras tareas.

6.2 La interfaz gráfica tendrá un menú inicial de *login*. Si la autenticación es correcta se pasará al menú principal de la herramienta.

6.3 El menú principal contendrá dos espacios vacíos del mismo tamaño en su zona central, el primero para insertar mensajes mediante *drag and drop*, y el segundo para mostrar información del progreso de la validación de los mensajes y la carga de datos. Debajo de los dos espacios habrá un botón para comenzar la carga de los mensajes y otro para detenerla.

6.4 En la zona inferior del menú habrá una pestaña para configurar la conexión. Mostrará las conexiones disponibles para el usuario que estén activas. También se mostrará la base de datos en la que se van a insertar los mensajes y su estado.

5 DISEÑO

5.1 Introducción

En este capítulo, de diseño del software, se realiza una descomposición del sistema en elementos que pueden implementarse por separado, describiendo cómo se organiza el sistema integrándolos a todos ellos (estructura global del sistema) y cómo se coordinan todos estos elementos.

El capítulo presentará un enfoque mixto, tanto arquitectónico como detallado, pues especifica tanto la arquitectura global del sistema (identificando módulos y sus relaciones) como cada una de sus partes, detallando los algoritmos y herramientas a emplear en cada una de ellas y la organización del código para comenzar la fase de implementación.

Las decisiones y planes acordados en esta fase de diseño tienen como fin satisfacer de la manera más completa posible los requisitos especificados en la fase anterior.

El documento presenta tres secciones principales, una de descripción de los aspectos generales del diseño, una de descripción detallada de cada componente, y una sección en la que se presenta el diagrama de bajo nivel, un diagrama de UML (Lenguaje Unificado de Modelado) que representa la arquitectura software del sistema. En el diagrama se muestra un enfoque inicial de las funciones principales de cada módulo del sistema, pero pueden desarrollarse actualizaciones del mismo debido a que surjan nuevas funciones auxiliares dentro de los módulos durante el proceso de implementación.

5.2 Aspectos generales del diseño

Para la implementación del sistema se empleará el IDE para Java NetBeans 7.4 pues contiene numerosas extensiones y librerías que facilitarán el cumplimiento de los requisitos impuestos además de ofrecer multitud de posibilidades en la interacción con bases de datos.

Para la base de datos se empleará el sistema gestor MySQL dado que es el sistema que se emplea en todas las bases de datos del CDM de EURECA e INTEGRATE, y además presenta una gran compatibilidad con NetBeans, que tiene incorporado un driver para crear conexiones JDBC con bases de datos en MySQL.

La herramienta a desarrollar no se tratará de un sistema autónomo y los usuarios necesitarán el uso de estas dos plataformas para el correcto funcionamiento del sistema. El objetivo de este proyecto no es la portabilidad, no se generará un fichero ejecutable que incluya todo el código del proyecto compilado porque no sería eficiente, y los usuarios deben establecer la conexión con una determinada base de datos dentro de su propio entorno en NetBeans. No se desarrollará como una aplicación de escritorio dado que no se poseen los conocimientos suficientes y requeriría aumentar considerablemente la fase de formación y tampoco se desarrollará como una aplicación web, web porque los datos con los que trabajará serán normalmente confidenciales.

El sistema presentará una conexión por defecto a una base de datos del CDM vacía (se desarrollará una función para vaciarla, comentada más adelante), cuyo nombre será "hl7_mysql_schema_2.6.2". Si los usuarios quieren probar la ejecución por defecto del sistema simplemente tendrían que crearse una base de datos local con dicho nombre que presente la estructura de la versión 2.6.2 del CDM. La URL de la conexión por defecto es la siguiente: `jdbc:mysql://localhost:3306/hl7_mysql_schema_2.6.2/?`

El sistema también soportará el uso de cualquier otra conexión (local o remota), y se desarrollarán las funciones necesarias para adaptar la ejecución a dicha conexión, pero es necesario que los usuarios generen dicha conexión en su entorno NetBeans.

En cualquiera de los dos casos, de ejecución por defecto o cambio de la conexión, la aplicación dará la opción al usuario de cambiar las propiedades de la conexión (usuario y contraseña de MySQL y URL de la conexión en el segundo caso).

El funcionamiento principal del sistema y el cumplimiento de la mayoría de requisitos se consigue gracias al uso de la tecnología Hibernate, que se trata de una herramienta de Mapeo objeto-relacional (ORM) para Java que facilita el mapeo (o traducción) de atributos entre una base de datos relacional y el modelo de objetos de una aplicación, mediante archivos declarativos XML o anotaciones en las clases Java de las entidades que permiten establecer estas relaciones. Permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información Hibernate le permite a la aplicación manipular los datos en la base de datos operando sobre objetos, con todas las características de la programación orientada a objetos. Hibernate convertirá los datos entre los tipos utilizados por Java y los definidos por

SQL. Hibernate genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias.

Para cumplir el requisito opcional de desarrollar una interfaz gráfica para la herramienta se empleará Swing, una biblioteca gráfica para Java.

5.3 Componentes

5.3.1 Módulo con la representación del CDM en objetos

Este componente del sistema contiene la implementación en lenguaje Java de todas las tablas del CDM. Para cada tabla se generan dos ficheros, un XML en el que se almacena para cada atributo la relación de tipos entre los que tenían en la base de datos original y los que se asignarán en el objeto Java que la represente, y una clase Java que contiene la representación de la tabla. Esta clase contiene todos los atributos de la entidad, métodos para obtenerlos y darles valor y varios métodos constructores, uno sin argumentos, otro con la clave primaria como argumento y otro con todos los atributos como argumento. En el caso de que la clave primaria de una entidad en la base de datos se componga de varios atributos se generará otra clase que contenga dichos atributos.

Los ficheros XML de *mapping* contienen un atributo denominado “catalog” cuyo valor será el nombre de la base de datos de la conexión por defecto, es decir, hl7_mysql_schema_2.6.2. La función que permite al usuario cambiar la conexión cambiará el valor de dicho atributo por el nombre de la base de datos que especifique el usuario.

Dentro del proyecto, todos estos ficheros se almacenarán en un paquete denominado CDM.

5.3.2 Módulo de validación e inserción de mensajes HL7 CDA

En este componente se encuentra la clase que contiene el programa principal de la aplicación, y que realiza todas las funciones requeridas. La clase se denominará ETL.java.

Dado que en la mayor parte de operaciones que realiza esta clase es necesario tratar con ficheros XML, se empleará JDOM, una biblioteca de código abierto para manipulaciones de datos XML optimizados para Java, que sirve tanto para validar ficheros XML como para actuar de descriptor de fichero de los mismos y optimizar su lectura en Java.

Contiene los siguientes atributos:

- Ruta del directorio de mensajes.
- Nombre de usuario de la base de datos.
- Contraseña de acceso a la base de datos.
- Nombre de la base de datos.
- URL de la conexión.
- Factoría de Hibernate: se construye a partir del fichero de configuración de la conexión, permite abrir una sesión para realizar operaciones en la base de datos.
- Sesión de Hibernate: permite la realización de transacciones con la base de datos como por ejemplo insertar, actualizar, borrar o consultar datos.

Contiene los siguientes métodos:

- Método para configurar la conexión (con dos opciones, utilizar la conexión por defecto cambiando la información de autenticación o cambiar la conexión al completo).
- Método para vaciar la base de datos.
- Método para validar un mensaje a partir de la ruta de dicho mensaje (comprueba tanto que el documento esté bien formado como que tenga toda la información obligatoria especificada en el estándar de intercambio de información clínica HL7 CDA).
- Método para cargar datos de un mensaje a partir de la ruta dicho mensaje.

Cabe destacar este último método, pues es el que realiza la funcionalidad principal del sistema. Extrae datos de un mensaje almacenando sus valores en los atributos de un objeto generado para la tabla que corresponda y hace uso del atributo de la sesión para guardar dichos objetos de forma persistente (método save() de la sesión de Hibernate, que sería el equivalente a realizar un Insert de SQL) o actualizar los campos de un objeto ya almacenado (método update() de la sesión).

En el programa principal se le presentarán todas las posibles opciones al usuario, se le solicitará que indique la ruta del directorio de mensajes mediante el explorador de carpetas y se le mostrará toda la información relativa a todas las etapas de la ejecución, ofreciéndole la posibilidad de salvar toda esta información en un fichero de log.

5.3.3 Base de datos del CDM

Base de datos que sigue modelo común de datos diseñado en los proyectos EURECA e INTEGRATE basándose en el estándar RIM de HL7 v3. El CDM contiene la información más relevante del RIM y presenta una gran capacidad para almacenar de forma común datos con representaciones muy variables, procedentes por ejemplo de ensayos clínicos o de historiales de hospitales. Este modelo puede experimentar actualizaciones adaptándose a nuevos datos clínicos recibidos que no puedan ser representados en el modelo, utilizando siempre como base para dichos cambios las especificaciones del RIM.

La organización principal del CDM es la siguiente: La información de pacientes (identificadores, información demográfica, etc…) u otras entidades (medicamentos, genes) se relaciona mediante roles de participación en diversos actos de ámbito clínico (como observaciones, tratamientos, procedimientos clínicos)

En las siguientes figuras se encuentran las últimas versiones tanto del RIM como del CDM.

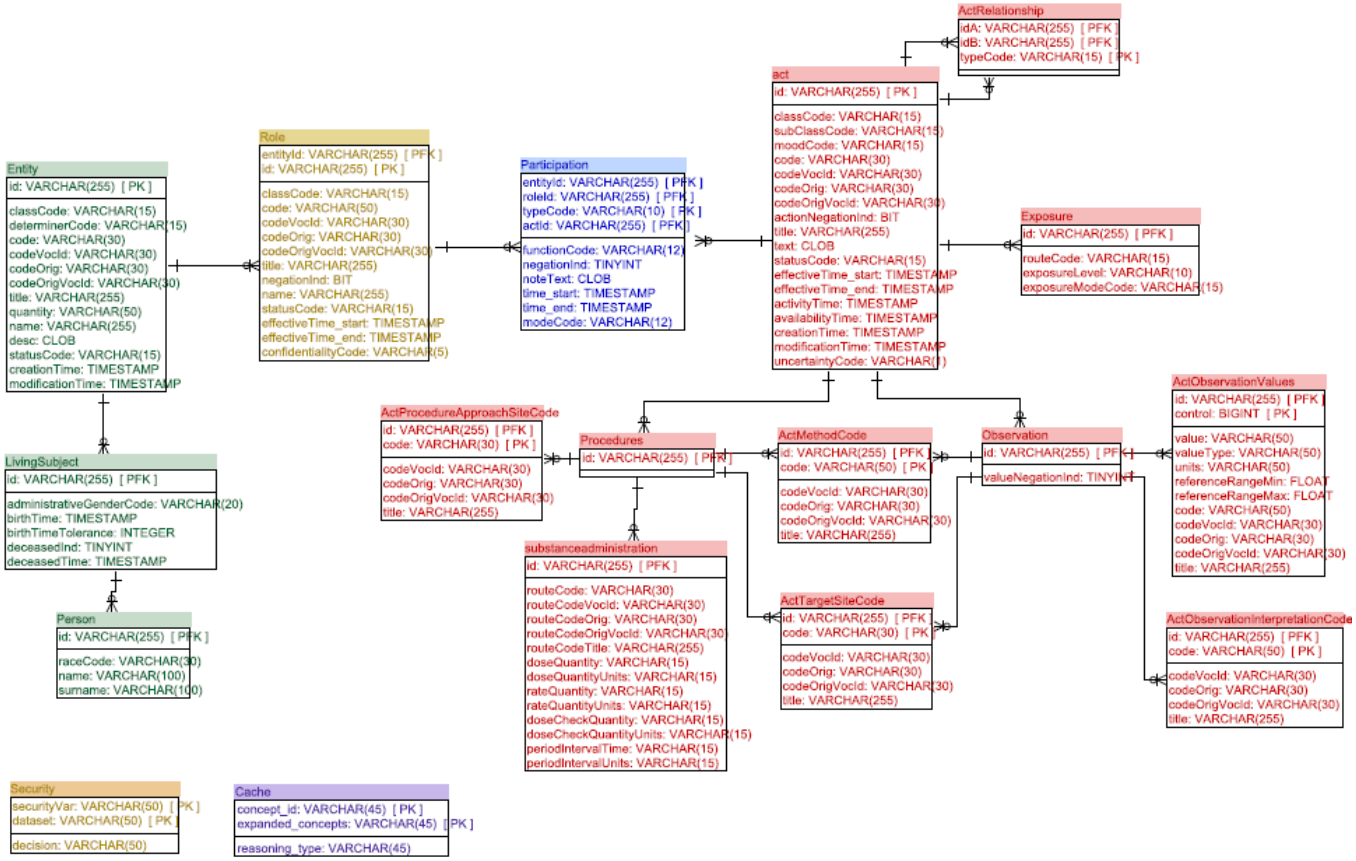


Figura 4. Versión 2.6.2 del modelo común de datos basado en el RIM de HL7 v3 de los proyectos EURECA e INTEGRATE.

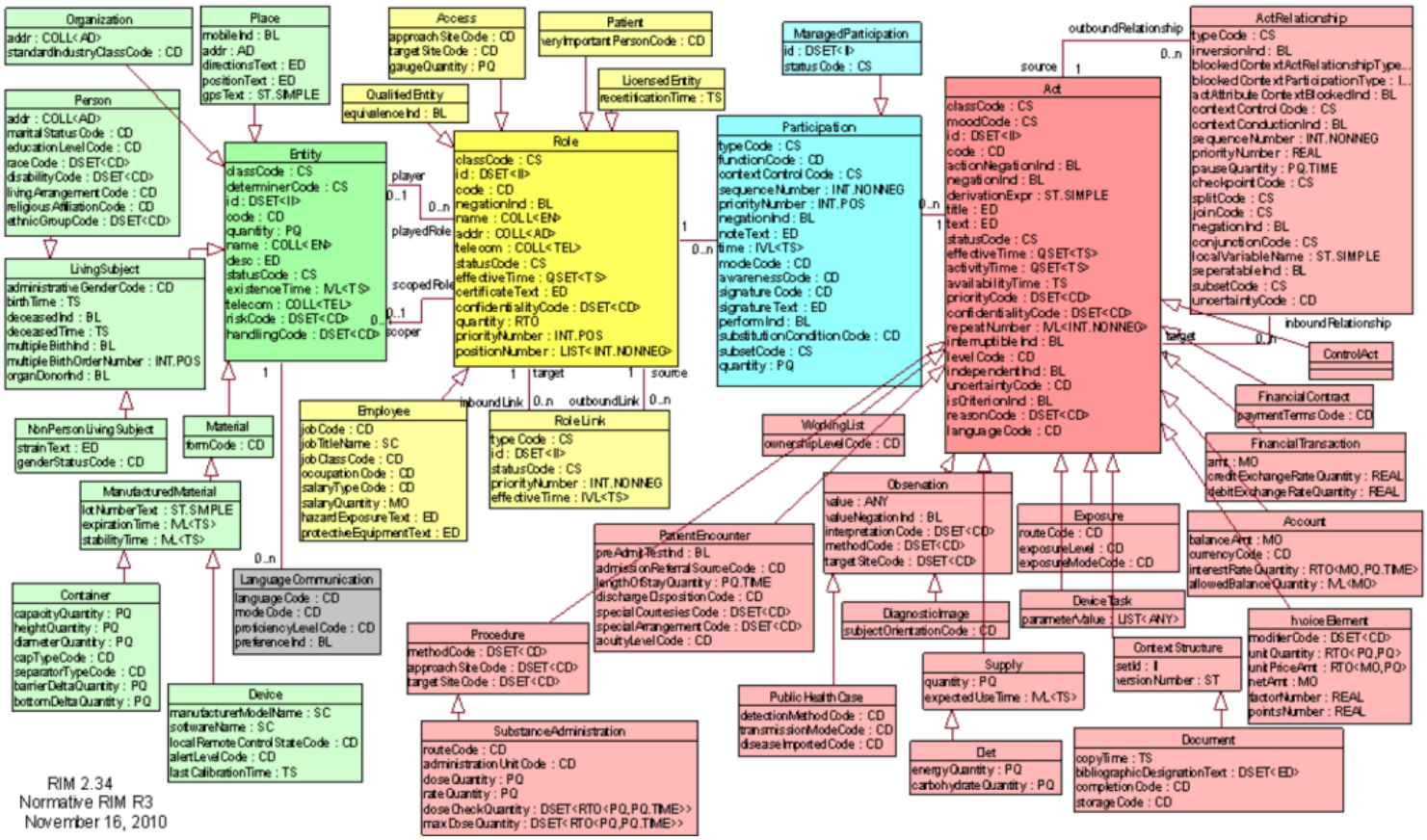


Figura 5. Versión 2.34 del modelo RIM completo del estándar HL7 v3.

Para el correcto funcionamiento del programa se deberá utilizar la versión 2.6.2 del CDM.

Este modelo fue desarrollado por Juan Manuel Moratilla Vargas, como Trabajo de Fin de Carrera para la Facultad de Informática de la UPM [20].

5.3.4 Conexión con la base de datos

Es un elemento fundamental para realizar las tareas principales de la aplicación. Permite tanto la generación de clases Java que representan a las entidades del CDM como la persistencia de los objetos generados de dichas clases que se generen a cuyos atributos se les darán valores extraídos de los mensajes.

Este componente genera dos ficheros XML: El fichero principal de configuración de Hibernate en el que se almacena toda la información relevante de la conexión, y el fichero de Reverse Engineering, en el que figuran los nombres de todas las tablas a mapear (se denomina reverso porque el uso común de Hibernate es el contrario, es

decir, a partir de objetos Java generar la estructura que tendrían sus clases en un modelo relacional de bases de datos).

Combinando ambos ficheros, Hibernate ofrece la posibilidad de generar de manera automática todas las clases que representen a las entidades que figuren en el fichero de Reverse Engineering y los *mappings* obtenidos tras comprobar los tipos de los atributos de dichas tablas mediante la conexión.

En el fichero de configuración (nombrado como hibernate.cfg.xml) se almacenan las siguientes propiedades obligatorias de la conexión: dialecto, clase del driver, que debe estar incluida en las librerías del proyecto y será siempre el driver de MySQL para Java (com.mysql.jdbc.Driver), usuario y contraseña de MySQL y URL de la conexión. También se pueden almacenar propiedades opcionales como mostrar por consola las consultas SQL en las que se traducen las acciones que realiza el programa. En este fichero también se almacena el nombre de los ficheros de *mapping* a los que tiene acceso la aplicación.

Para ofrecer la posibilidad al usuario de editar las propiedades de la conexión se desarrollará en la clase principal ETL una función que realiza cambios en este fichero.

El fichero de Reverse Engineering contiene un atributo “match-catalog” cuyo valor será el nombre de la base de datos de la conexión por defecto, es decir, hl7_mysql_schema_2.6.2. La función que permite al usuario cambiar la conexión cambiará el valor de dicho atributo por el nombre de la base de datos que especifique el cliente.

5.4 Diagrama de bajo nivel

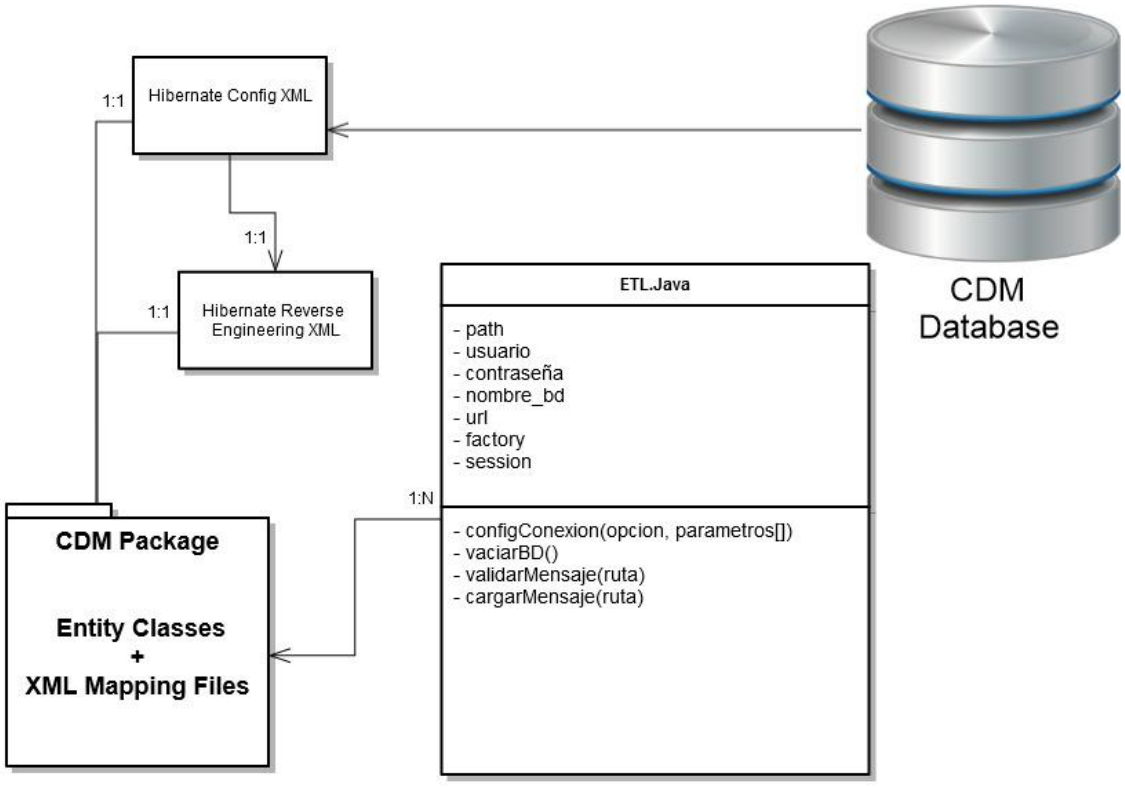


Figura 6. Diagrama UML que engloba a todos los ficheros de los que se compone la herramienta a desarrollar.

6 DESARROLLO

6.1 Introducción

En este capítulo se documenta detalladamente la fase principal y de mayor duración del proyecto, la de desarrollo del software. Esta fase, además de la implementación, incluye la validación y depuración parcial de los componentes según se han ido desarrollando mediante pruebas unitarias de los mismos o pruebas de integración con otros componentes ya desarrollados, para comprobar si cumplen con el funcionamiento deseado.

El capítulo consta de una subsección explicativa de los cambios realizados con respecto a las fases previas de análisis y diseño, y de un capítulo en el que se describen la metodología, procedimientos seguidos y algoritmos empleados para implementar cada componente de la herramienta. Finalmente contiene un apartado que muestra la organización de los ficheros dentro de la herramienta, el formato de entrega y cómo se

realizaría una ejecución básica de prueba, indicando todos los pasos que se llevan a cabo durante una ejecución, interactuando con el usuario.

6.2 Discrepancias con respecto al diseño y requisitos

Dado que las etapas de diseño y análisis de requisitos son previas a la etapa de codificación o desarrollo, durante la realización de esta han surgido ciertas incompatibilidades con lo expuesto en dichas fases que son relevantes de analizar.

En primer lugar, se presentan ciertas discrepancias con algunos requisitos. No se han cumplido algunos de ellos debido a que no se consideran lógicos, con la intención de mejorar la eficiencia de la herramienta, o porque no se consideran relevantes para satisfacer los objetivos principales de este proyecto, o en algunos casos simplemente se ha añadido funcionalidad. A continuación se exponen los requisitos que han sido revisados o rechazados:

Requisitos	**Revisión**
2.1: “Si selecciona un fichero en lugar de un directorio se mostrará un mensaje de error, pero se le permitirá al usuario volver a seleccionar el directorio.”	Se ha implementado la inserción de la ruta del directorio que contiene los mensajes HL7 accediendo al explorador de carpetas mediante la clase JFileChooser de Java, que da la opción de permitir únicamente seleccionar directorios. Los ficheros no aparecen, y obviamente no es posible seleccionar un directorio que no exista, no hay posibilidad de error.
4.2: “A continuación se le permitirá seleccionar la ruta del directorio que contenga los mensajes a procesar, si no es correcta (no existe, no es un directorio), se mostrará un mensaje informando al usuario del error, y se le permitirá volver a seleccionar el directorio”	
1.3: “Se deberá mantener un fichero XML con información relativa a las propiedades de la conexión”	A dicho fichero se le añade la propiedad “hibernate.default_schema” (que determinará la base de datos en la que se carguen los mensajes) además de las mencionadas en el requisito como los datos de autenticación, URL de la conexión, etc…
2.5: “La herramienta sólo aceptará el formato de fechas especificadas por el	Debe aceptar fechas vacías, por tanto la única restricción es que estén en un

estándar, es decir, que los valores de todas las etiquetas effectiveTime deberán tener el siguiente formato: YYYYMMDD, sin ningún delimitador.”	formato que sea posible de convertir al tipo datetime de MySQL. A la hora de validar los mensajes según el estándar no se impondrá ninguna restricción en las fechas. Y en el mapping a objetos Java se permiten tanto instancias de la clase date (en desuso, pero que se puede emplear para obtener el tiempo actual mediante su constructor y se realiza correctamente su conversión al tipo datetime) como cadenas de texto con diversos formatos de fecha.
3.9: “Una vez cargada la información de todos los mensajes, se generará un acto que represente al estudio del que se han extraído los datos, con valor “CLNTRL” (*Clinical Trial*) en el atributo *classCode*.”	Se le tendrá que pedir al usuario el nombre del ensayo clínico, y relacionarlo con todos los actos con classCode=”FILE” representativos de cada paciente.
4.1: “En primer lugar, al iniciar la ejecución se mostrará un mensaje indicando al usuario que se autentifique, respondiendo si los parámetros insertados son correctos o volviendo a solicitarle que los inserte si no son correctos.”	No tiene sentido, no es una aplicación centralizada, es propia para cada usuario, que puede configurar su conexión como desee sin un tercero que realice un control de acceso. Ya que dicha conexión depende su configuración de MySQL.
5.1: “Se le pedirá al usuario introducir el nombre de usuario y contraseña de acceso a la base de datos y se compararán los valores introducidos con los parámetros de la conexión. Si no coinciden, se mostrará un mensaje de error y finalizará la ejecución.”	
4.3: “Se mostrará información del progreso de la validación de los mensajes, indicando las líneas que provoquen error y si no se produce, un mensaje en el que se especifique el nombre del fichero indicando que se ha procesado correctamente”	Para no recargar la información mostrada durante la ejecución y el fichero de log simplemente se mostrará el número de mensajes sin error y el total de mensajes analizados.
4.6: “Se irá informando al usuario del progreso de la carga de cada mensaje, y al finalizar se mostrará un mensaje en el que se especifique el nombre del fichero indicando que se ha procesado	En el fichero de log se mostrará el nombre de cada fichero a la hora de cargar su contenido y todos los objetos que se generen al procesarlo, con los valores de todos sus atributos.

correctamente. Al finalizar la carga se mostrará el número de mensajes cargados y el número de mensajes con avisos.”	
Feature 6: Interfaz gráfica	No se ha implementado el requisito opcional de la interfaz gráfica para la herramienta. Aumentaría la comodidad de las ejecuciones, pero no es el tipo de aplicación que más la requiere. La herramienta sí tiene un componente gráfico, se ha utilizado la implementación de demo de Oracle de la clase JFileChooser de la librería Java Swing para dar la opción al usuario de seleccionar el directorio de mensajes.

Tabla 1. Revisiones de los requisitos surgidas durante la fase de implementación.

Con respecto al diseño, tal y como se comentó en el capítulo anterior, se establecía la estructura global del sistema, pero se daba cabida a futuras actualizaciones durante la implementación de la herramienta. Han surgido numerosas funciones y atributos adicionales, con propósito auxiliar, para facilitar la implementación de las funcionalidades principales de la herramienta, especialmente en la clase principal (ETL.java). Se explicarán en detalle en la siguiente subsección de descripción de la implementación de cada componente.

En dicha sección se comentaba que los usuarios necesitaban el uso de MySQL y NetBeans para el correcto funcionamiento del sistema. Pues bien, se ha conseguido abstraer la herramienta del uso de NetBeans creando un JAR con todos los ficheros que componen la herramienta, por lo que se ha ganado mucha portabilidad, y la herramienta ha pasado a depender de una única plataforma, MySQL.

Las conexiones que se configuren en el fichero de propiedades de Hibernate no se requiere que estén creadas previamente en el entorno de NetBeans. Se ha cambiado la conexión por defecto por una que no tenga ningún esquema asociado en su URL, simplemente la máquina local (`jdbc:mysql://localhost:3306/?zeroDateTimeBehavior=convertToNull`). El usuario podrá configurar el esquema al que acceder, estando por defecto asociado a un esquema con nombre “cdm”, en un nuevo atributo añadido al fichero de configuración ("hibernate.default_schema"), en lugar de en la propia URL de la conexión. El nombre pensado en un principio como esquema por defecto, ““hl7_mysql_schema_2.6.2”, no es compatible con Hibernate porque no acepta números en los nombres de las bases de datos ya que provoca un error en las consultas SQL que se generan.

El haber añadido el atributo del esquema por defecto en el fichero de configuración de Hibernate también ha simplificado la implementación del método de configurar la conexión, ya que se ha eliminado el atributo "catalog" en los ficheros de *mapping*, de modo que para cambiar la base de datos de destino de los objetos generados simplemente hay que cambiar el valor de dicho atributo del fichero de configuración en lugar del atributo "catalog" de todos los ficheros de *mapping*. Además, no fue posible implementar el método de configuración de la conexión cambiando el atributo en todos los ficheros de *mapping*, pues se requería finalizar la ejecución y que se realizara una nueva para que se cargara el cambio.

Tampoco es necesario cambiar el atributo "match-catalog" del fichero de Reverse Engineering. Este fichero únicamente es necesario para la generación automática de las clases Java y los ficheros de *mapping* a partir de las tablas de una base de datos accesible por medio de una conexión JDBC, y deja registrado en un atributo la base de datos a partir de la cual ha generado dichos ficheros. Pero una vez ha generados, no se requiere acceder al contenido de dicho fichero nunca más durante las ejecuciones.

En caso de que el usuario quiera realizar una ejecución por defecto, cambiando los mínimos parámetros posibles de configuración de la conexión simplemente deberá crearse una base de datos con el modelo 2.6.2 del CDM (mediante el script entregado junto a los ficheros de implementación de la herramienta) en su máquina local, nombrarla como "cdm" y cambiar únicamente la contraseña de la conexión, asignando la que haya establecido para su usuario root de MySQL.

El uso de un base de datos de MySQL como destino de almacenamiento de los mensajes HL7 es obligatorio, ya que aunque se podría configurar el fichero de propiedades para emplear una conexión a otro sistema gestor de bases de datos sería necesario incluir el driver para dicho sistema en el JAR.

6.3 Análisis detallado de la implementación de cada componente

En esta sección se describen los detalles de implementación de todos los componentes que constituyen la herramienta, en el orden en el que han sido implementados:

6.3.1 Base de datos

Se ha empleado el script de SQL de generación de bases de datos vacías de la versión 2.6.2 del CDM, para crear la base de datos en la máquina local en MySQL, nombrándola como "cdm".

Ha sido necesario alterar una de las tablas del CDM desde MySQL antes de generar el módulo con su representación en objetos Java, ya que presentaba una incompatibilidad con Hibernate. En concreto ha sido originada por el campo “desc” de la tabla Entity, cuya función es de almacenar una descripción textual o multimedia de la instancia de la entidad. El nombre escogido para este campo es también una palabra reservada de MySQL, y provoca conflicto en todas las consultas (tanto de inserciones, selección, actualizaciones) que incluyan a dicho campo (o en el caso de Hibernate siempre que se realice una consulta a la tabla Entity, dado que en las consultas SQL que genera, a partir del código Java, siempre se incluyen todos los campos de la tabla consultada). Se ha sustituido su nombre por “descc” para solucionar el problema [44].

También se han importado en la máquina local bases de datos del CDM con contenido, mediante dumps, exportaciones de bases de datos en scripts de SQL como copia de seguridad (scripts como el empleado para generar el CDM vacío que además contienen sentencias INSERT con el contenido de la base de datos a exportar). Estos dumps contienen la información de ensayos clínicos empleados en los proyectos EURECA e INTEGRATE. Se han empleado estas bases de datos para realizar comprobaciones de los posibles valores que toman los campos de las tablas del CDM, como soporte a la implementación para ver cómo se organiza la información, y en algún caso como prueba para ver cómo se comporta la herramienta al tratar de insertar información en una base de datos que no esté vacía.

6.3.2 Ficheros de configuración de Hibernate

Estos ficheros se han generado automáticamente desde NetBeans, creando previamente una conexión JDBC para el esquema generado en el paso anterior, esta conexión servirá de apoyo para generar los ficheros de configuración y la representación del CDM en clases Java, pero no tiene porqué utilizarse para ejecuciones posteriores. En la siguiente captura se muestran el esquema y la conexión generada a partir de él desde NetBeans:

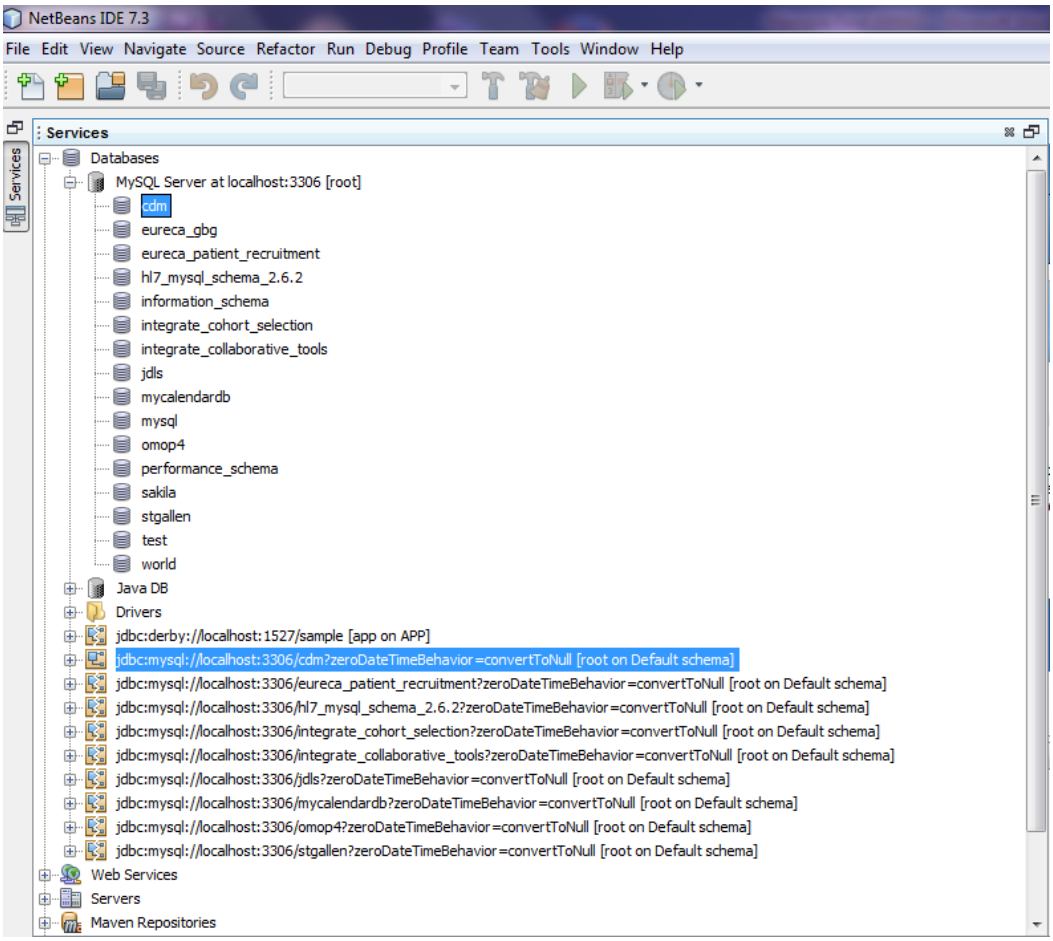


Figura 7. Acceso y conexión desde NetBeans a la base de datos del CDM creada previamente.

A continuación, NetBeans permite la opción de crear un proyecto Java a partir de dicha conexión, mediante la inclusión del framework de Hibernate.

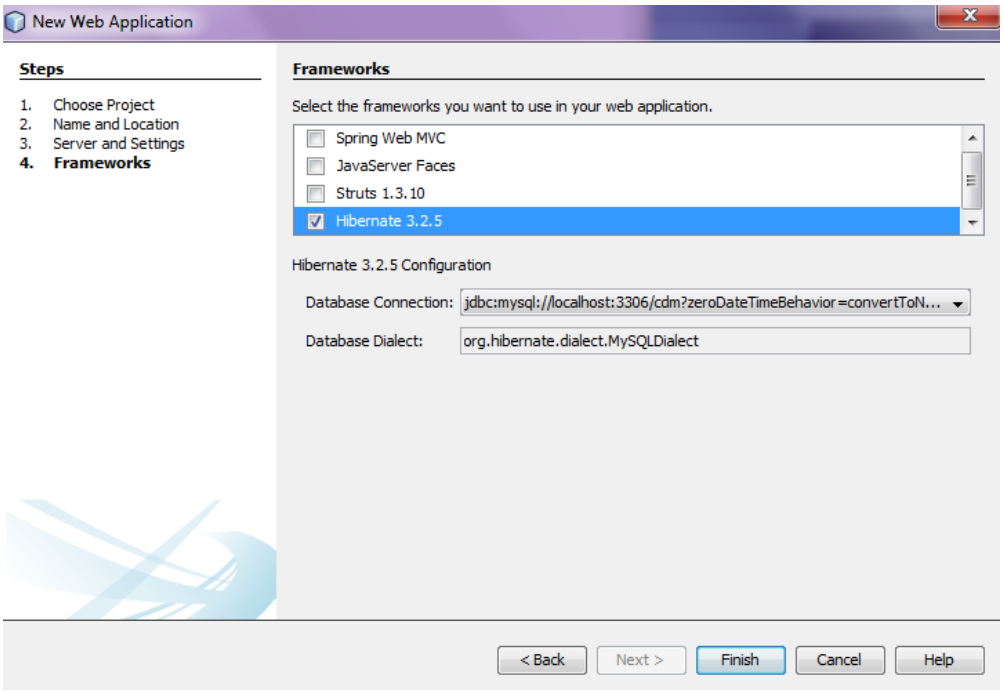


Figura 8. Creación del proyecto Java con el framework de Hibernate, a partir de la conexión creada anteriormente.

Al crearse el proyecto se generará automáticamente el fichero XML de propiedades de Hibernate (hibernate.cfg.xml), extrayendo los valores de sus atributos de la conexión. En un principio el contenido del fichero únicamente incluirá las propiedades básicas de la conexión, es decir, los siguientes atributos con los siguientes valores:

- `hibernate.dialect = org.hibernate.dialect.MySQLDialect` (no debe cambiar).
- `hibernate.connection.driver_class = com.mysql.jdbc.Driver` (no debe cambiar).
- `hibernate.connection.url = jdbc:mysql://localhost:3306/cdm?zeroDateTimeBehavior=convertToNull` (sólo cambiará en accesos a máquinas remotas).
- `hibernate.connection.username = root`, por defecto.
- `hibernate.connection.password = *`

La información que representan estas propiedades se encuentra en la sección correspondiente a este componente de la fase de diseño, 5.3.4, y en el requisito 1.3 en el que se expone cómo se debe establecer la representación del CDM en Java.

Pero el fichero soporta la adición de gran cantidad de información y propiedades, como el nombre de la base de datos por defecto sobre la que se realizarán operaciones, la ubicación de todos los ficheros de *mapping* que se emplearán, o las propiedades opcionales de mostrar por consola el código SQL que se genere, la opción de generar estadísticas, etc…

La función de este fichero es la creación de un objeto de la clase Session de Hibernate, que será el encargado de realizar toda la interacción con la base de datos, con métodos para guardar los objetos generados en ella de forma persistente, o para realizar consultas, actualizaciones, eliminaciones, etc…Se explicará en más detalle en la sección correspondiente a la implementación de la clase principal, 6.3.4.

En el contenido final del fichero se ha eliminado el nombre de la base de datos dentro de la propiedad de la URL, quedando una ruta únicamente a la máquina local y se ha añadido la propiedad ”hibernate.default_schema” para que sea el único punto en el que se determine la base de datos a la que se va a acceder.

A continuación se generará el fichero de Reverse Engineering (hibernate.reveng.xml). Para ello, se ha empleado la opción de “Hibernate reverse engineering wizard” que proporciona NetBeans a la hora de crear ficheros dentro de un proyecto. Se hace uso de la conexión registrada en el fichero de configuración para encontrar las tablas disponibles.

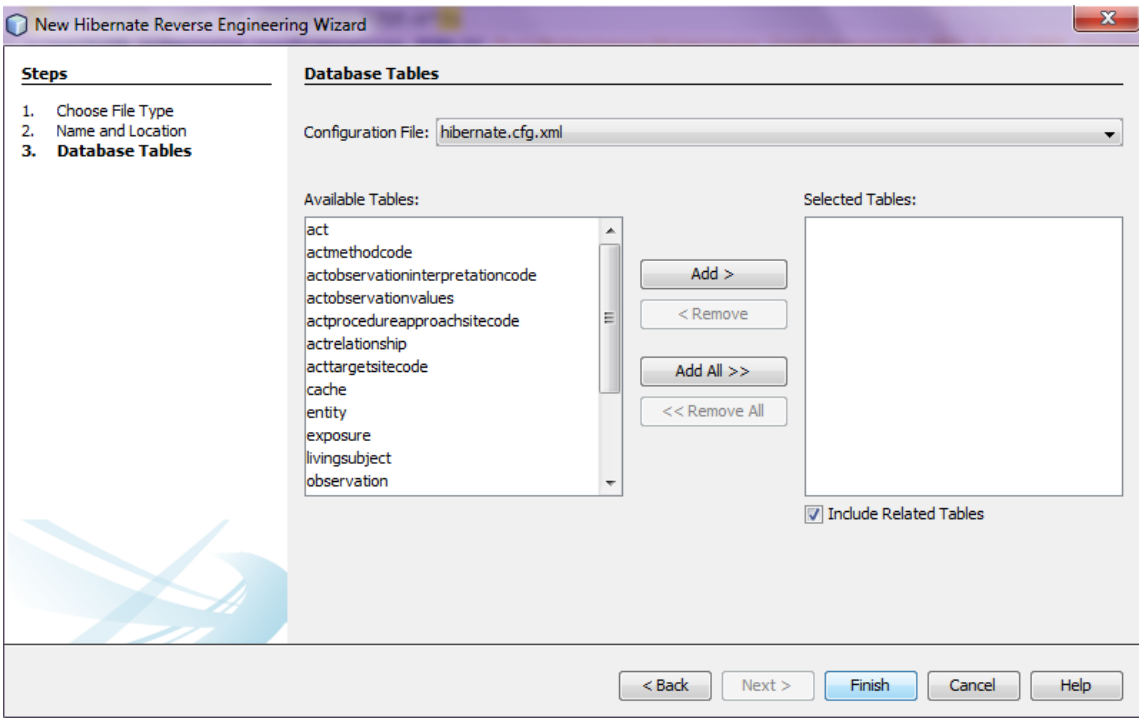


Figura 9. Creación del fichero Reverse Engineering en NetBeans, hallando todas las tablas accesibles desde la conexión.

Una vez creado el fichero se registrará en él los nombres de todas las tablas seleccionadas.

Y finalmente, se utiliza la opción de NetBeans de "Hibernate mapping files and POJOs from database" para combinarlos dos ficheros generados anteriormente y generar una versión preliminar de la representación de la base de datos en clases Java (POJOs, Plain Old Java Object), y ficheros XML de *mapping* para cada una de ellas.

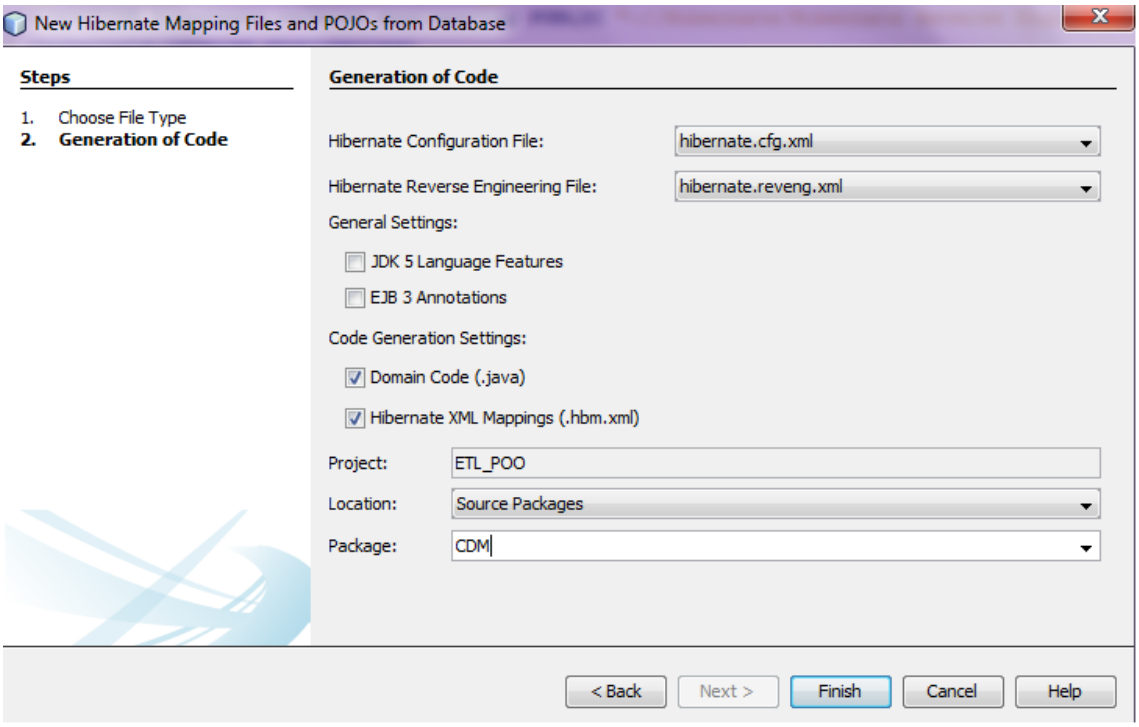


Figura 10. Generación de las clases Java y ficheros de mapping para las tablas del CDM.

6.3.3 Módulo con la representación del CDM en objetos

La generación de éste módulo se explica en el apartado anterior, pero una vez creando han tenido que realizarse numerosas modificaciones.

Como se ha comentado en anteriores ocasiones, este módulo (paquete CDM dentro del proyecto) contiene clases Java representativas de cada tabla del CDM, en algunos casos con más de una clase para representar una tabla, ya que para aquellas tablas cuya clave primaria esté compuesta por más de un campo se crea una clase adicional para el identificador de dicha tabla, que será un atributo dentro de la clase principal que represente a la tabla. Esta es la solución que aporta Hibernate para, agrupando la clave primaria en un único atributo, poder nombrar siempre a la clave primaria completa de

cada clase como un único atributo "id" habilitando un método constructor únicamente con dicho atributo como parámetro.

Las clases contienen todos los atributos representativos de los campos de su tabla del CDM asociada con su tipo asignado en el *mapping*, con métodos set(valor) y get() para darles valor u obtener el valor que tengan, 3 métodos constructores, uno sin argumentos, otro con los argumentos que tengan la opción "Not Null" en la base de datos, y otro con todos los atributos de la clase como argumentos, y un método toString() añadido, que se describirá posteriormente.

Y el módulo contiene también ficheros XML de *mapping* para cada tabla, describiendo cada campo a qué atributo se mapeará y con qué propiedades (tipo de datos de Java, longitud, etc…).

A continuación se enumeran y detallan los cambios realizados en el módulo generado por defecto en el apartado anterior:

- Todos los atributos relacionados con fechas (como por ejemplo el effectiveTime de los actos, el birthTime de LivingSubject, o el creationTime de múltiples clases) se mapean por defecto a la clase Date de Java. Esto supone un problema debido a que dicha clase está en desuso. Al extraer los datos de fechas de los mensajes HL7 como texto y crear un objeto Date a partir de dichos valores, no se almacenan correctamente y al persistir ese objeto queda con valores alterados. Por tanto, tras realizar varias pruebas, la solución más adecuada que se ha encontrado es mapear los atributos de fechas como String, ya que, si tienen un formato correcto, no hay problemas al persistir los objetos y MySQL acepta los valores aunque los campos en las tablas sean del tipo date o datetime.
  El caso del atributo creationTime es excepcional. Porque, pese a que la clase Date esté en desuso, su constructor por defecto sin argumentos no lo está, y es la manera más cómoda de obtener la fecha actual. Por tanto, ya que no se necesita dar un valor determinado ni usar los métodos problemáticos de la clase Date, los atributos de creationTime permanecen con el tipo Date.

- La tabla ActObservationValue presenta un campo denominado "control". Este atributo surge debido a que la clave primaria de esta clase en versiones anteriores del CDM estaba compuesta por el identificador (mismo que el del acto al que esté asociado) y el valor de la observación. Pero como un acto de observación puede tener varios valores asociados (razón por la que los valores de observación están en una tabla aparte), y podría darse la situación de que un mismo acto tenga asociados dos valores iguales pero, por ejemplo, registrados en distinta fecha, dado que la fecha no forma parte de la clave primaria, ambos valores serían indistinguibles, y no se podría almacenar uno de ellos porque no se puede almacenar dos entradas de una tabla con la misma clave. De modo que se originó un atributo autoincremental denominado control, y se añadió a la clave primaria, quitando de ella el valor. Así, por cada entrada que se registre en

la tabla ActObservationValue se aumentará el valor de control y todas las entradas tendrán un valor distinto para este campo.
Para que se asigne un valor autoincremental de dicho campo, en función de las entradas que ya tenga registradas la tabla, en MySQL simplemente hay que realizar las inserciones sin especificar ningún valor ni incluir en la consulta de inserción dicho campo.
El problema surge al hacer el *mapping* de esta tabla, que incluye el atributo control en la clase correspondiente a la clave primaria de ActObservationValue. Inicialmente se mantenía una variable entera en la clase principal que se incrementara con cada inserción de un observationvalue, pero si la base de datos ya tenía contenido antes de la ejecución, o se desearan realizar varias ejecuciones, esta variable perdería utilidad, y habría que realizar consultas Select en la base de datos antes de asignar un valor a un nuevo objeto de esta clase para que el valor del control sea el correcto. Dado que esto aumenta la complejidad y entorpece la ejecución se ha optado por eliminar dicho campo del *mapping*, ya que al ser autoincremental no es necesario que aparezca en las consultas.

- Se ha añadido un método toString() en todas las clases, para poder incluir la información de los objetos que se guarden y los valores de sus atributos en el fichero de log que registra toda la información de la ejecución de la herramienta. Se ha añadido el método automáticamente desde NetBeans en todas las clases, pero modificándolo para que en el caso en el que el identificador de la clase sea un objeto de otra clase (clave primaria múltiple), se llame al método toString() de la clase del identificador.

6.3.4 Módulo principal: clase que implementa las funcionalidades requeridas y ficheros asociados

En esta subsección se describe el núcleo de la herramienta. El paquete ETL del proyecto constituye la parte central del proyecto, especialmente la clase ETL.java, que contiene el punto de acceso de la herramienta, el método main del fichero JAR principal que constituye la herramienta, así como la implementación de la mayor parte de la funcionalidad requerida.

Este módulo incorpora además la clase FileChooserDemo.java, no implementada por el alumno íntegramente, extraída de un ejemplo de la página de documentación y manuales de Oracle [32]. Esta clase hace uso de la clase JFileChooser de Java, implementación gráfica de la biblioteca Swing para un selector de ficheros. Se ha usado para permitir al usuario seleccionar el directorio donde estén los mensajes que desea cargar implementando gráficamente una llamada al explorador de carpetas.

Se han realizado algunos cambios en el ejemplo extraído de la documentación de Oracle para adaptar la clase al funcionamiento deseado, descritos a continuación:

- En primer lugar, se ha establecido un atributo String filepath, para almacenar la ruta completa que seleccione el usuario, que es el propósito por el cual se usa esta clase.

- Dentro del constructor de la clase, se ha empleado el método `setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY)` para permitir al usuario seleccionar únicamente directorios.

- Se ha eliminado todo el código relativo a guardar ficheros, ya que el ejemplo también incorporaba esa opción.

- En el método que implementa la acción consecutiva a pulsar el botón "open" al seleccionar un directorio, se ha añadido que se dé valor al atributo filepath comentado anteriormente, con la ruta absoluta del directorio que seleccione el usuario.

- En el método createAndShowGUI() que se ejecuta desde el main de esta clase para crear y mostrar la interfaz gráfica de Java se cambiado la opción "EXIT_ON_CLOSE" del marco por "HIDE_ON_CLOSE". Esto es debido a que el main de esta clase se invocará desde el main de la clase principal (ETL.java), y era necesario cambiar esta opción para que no finalice el main de la clase principal si se cierra la interfaz, ya que la clase principal realiza numerosas operaciones tras haber seleccionado el directorio y no debe finalizar su ejecución. Desde la clase principal se genera un objeto de esta clase, FileChooserDemo, cuando llega el momento de seleccionar el directorio. Y posteriormente da valor null al objeto creado de esta clase para que el recolector de basura de Java elimine el proceso del main de esta clase.

El módulo también contiene un icono que forma parte de la interfaz.

A continuación se describe el contenido completo de la clase ETL.java, analizando sus atributos y métodos. Presenta los siguientes atributos, que difieren de los establecidos en la fase de diseño, ya que contenía una primera aproximación y finalmente algunos de ellos no se han considerado necesarios:

- nombre_bd: Nombre de la base de datos que está registrado en el fichero de configuración de Hibernate en el momento previo a la carga de mensajes (se lee el fichero de configuración en dicho punto de la ejecución para dar valor a esta variable). Se establece como atributo para poder tener disponible en todo momento su valor, por ejemplo para informar desde el main de que la base de

datos con dicho nombre no está vacía, o informar desde el método de carga de mensajes de en qué base de datos se van a realizar las inserciones.

- factory: Atributo de tipo SessionFactory, clase de Hibernate encargada de crear instancias de sesiones. Se construye a partir del fichero de configuración de la conexión, permite establecer una sesión para realizar operaciones en la base de datos.

- session: Atributo de tipo Session, clase de Hibernate que permite la realización de transacciones con la base de datos como por ejemplo insertar, actualizar, borrar o consultar datos a través de instancias de las clases que mapean las tablas de la base de datos. Es la clase central que representa la API de persistencia proporcionada por Hibernate. Sus métodos principales son save, update, delete y createCriteria, que generan consultas SQL de los siguientes tipos INSERT, UPDATE, DELETE y SELECT respectivamente.

- log: Atributo de texto que va almacenando toda la información relevante de la ejecución mediante concatenaciones desde la mayor parte de métodos de la clase. Al finalizar la ejecución se da la opción al usuario de exportar el contenido de este atributo en un fichero de texto. Este fichero contendrá:

    - Ruta absoluta del directorio seleccionado que contenga los mensajes HL7 a insertar.

    - Estadísticas de validación de los mensajes en función de las reglas de formación de documentos XML y en función del estándar HL7 CDA. Se indica el número de mensajes con errores de formación o validación y el número total de mensajes analizados y se describen los errores concretos especificando el motivo y la línea y el fichero en el que se detectan. También se muestran avisos si el directorio contiene otros directorios anidados o ficheros con extensión distinta a XML, indicando que la herramienta no los puede procesar.

    - Descripción de las inserciones de los datos, indicando los ficheros que se procesan y todos los objetos generados a partir de ellos, mostrando los valores que toman sus atributos. Se muestran avisos si hay conceptos de terminologías desconocidas, y al final el número total de avisos.

    - Tiempo total de la ejecución, en minutos.

En las posteriores subsecciones se describe la funcionalidad (y las técnicas mediantes las cuales se ha implementado esta funcionalidad) de cada uno de los métodos de la clase principal ETL.java.

6.3.4.1 comprobarVoc

Función auxiliar utilizada en la carga de datos. Recibe como argumento una cadena de texto que debe ser un código OID identificativo de un vocabulario y devuelve un valor booleano falso si el código difiere de los códigos de SNOMED-CT (2.16.840.1.113883.6.96), LOINC (2.16.840.1.113883.6.1) o HGNC (2.16.840.1.113883.6.281), lo que indicará que el concepto estará anotado en una terminología que no pertenece al Core Dataset de los proyectos EURECA e INTEGRATE.

Se llama a este método usando como argumento el código de vocabulario de cada concepto que se vaya a cargar en el método de carga de datos, y si devuelve falso concatena un aviso de terminología desconocida en el atributo log.

El método de carga de datos lleva un contador del número de avisos de terminología que se van produciendo, imprimiendo su valor final en el atributo de log.

6.3.4.2 configConexion

Función que recibe como argumentos los parámetros de la conexión que el usuario desea cambiar, recogidos a través de la entrada de datos. Si el usuario desea mantener un parámetro con su valor no introducirá ningún valor para ese parámetro y se le pasará a este método como null. Toma como argumento una variable booleana que tomará valor false cuando el usuario desee realizar una ejecución local (valor por defecto de la variable que se mantendrá si el usuario no introduce ningún valor cuando se pida que decida entre establecer una conexión local o remota) y valor true cuando se vaya a establecer una conexión con una base de datos remota. También recibe como argumentos 4 variables de texto, el nombre de la base de datos, usuario y contraseña de MySQL y la dirección de la máquina remota.

En este método en primer lugar se lleva a cabo una comprobación de que los parámetros introducidos sean correctos, es decir, comprueba el único posible error que sería cuando el usuario decida establecer una conexión remota pero no haya introducido la dirección del host al que desea acceder. En ese caso el método finaliza y se utilizarán los valores por defecto que presente la conexión.

A continuación se hace uso de la clase SAXBuilder de la librería JDOM para crear un descriptor de fichero de lectura especializado para XML para poder procesar el fichero de configuración de Hibernate, obteniendo un objeto de la clase Document de JDOM, que contiene todo el fichero XML. Se parte del elemento raíz del fichero y se almacenan sus elementos "property" en una lista. Se recorre esta lista cambiando los parámetros que corresponda (los que no tengan valor null).

Finalmente se utiliza un descriptor de fichero de escritura (FileWritter) para guardar los cambios en el documento, haciendo uso previamente de la clase XMLOutputter de JDOM para producir un XML de salida a partir del objeto Document que ha sido modificado, que se le enviará como parámetro al método write del descriptor de fichero.

Con la implementación de este método se demuestra que Hibernate ofrece la posibilidad de cambiar en tiempo de ejecución el esquema al que se va a acceder [45].

6.3.4.3 comprobarBD y vaciarBD

Métodos implementados para comprobar en primer lugar si la base de datos a la que se pretende acceder no está vacía y posteriormente dar la posibilidad al usuario de que vacíe el contenido de dicha base de datos.

El método comprobarBD implementa una consulta SELECT a la tabla de actos de la base de datos de destino. Es suficiente con analizar la tabla de actos, es imposible que una base de datos del CDM contenga entradas en otras tablas y tenga vacía la tabla de actos. Las primeras inserciones que se hacen son las de los actos de los ficheros que representan a los mensajes y la del acto que representa al ensayo clínico.

Para realizar este SELECT se emplea el método createCriteria del atributo de la sesión, pasándole como argumento "CDM.Act.class". El objeto Criteria (de Hibernate) que devuelve contendrá una lista con todos los actos registrados en la base de datos. La comprobación de si dicha lista está vacía determina el valor de la variable de retorno de esta función. Si no está vacía se devolverá true y si lo está, false.

El método vaciarBD, al que sólo se accede si la base de datos no está vacía y el usuario decide vaciarla, sigue el mismo procedimiento que el método anterior, creando listas a partir de objetos Criteria para cada una de las tablas de la base de datos. Después se recorre el contenido de dichas lista (con bucles "for each", con una variable interna que en cada iteración toma como valor un elemento de la lista) eliminando cada elemento de cada lista, con el método delete de la sesión.

6.3.4.4 validarMensajes

Método que recibe como argumento la ruta del directorio que contenga los mensajes a cargar e implementa la funcionalidad de analizar dichos mensajes, detectando errores de formación y/o validación.

Para detectar estos errores hace uso de la clase SAXBuilder. Creando una instancia de dicha clase inicializándola en primer lugar sin argumentos, para detectar errores de construcción de ficheros XML (como etiquetas sin cierre por ejemplo), y en segundo lugar inicializándola con un descriptor de fichero del XSD desarrollado como implementación de las reglas y estructuras establecidas por el estándar HL7 CDA (explicado en la sección 6.3.5).

Se realiza un bucle para recorrer todos los ficheros que contenga el directorio pasado como argumento, en primer lugar comprobando que no sean otros directorios o que no tengan una extensión distinta a ".xml". Si se da alguno de estos casos se indicará que la herramienta los ignorará y se aumentará un contador de ficheros ignorados.

Para los ficheros XML bastará con enviar como argumento un descriptor de fichero (instancia de la clase File) de los mismos al método builder de las instancias de SAXBuilder comentadas anteriormente. En primer lugar se realizará este paso con la instancia inicializada sin argumentos, procesando todos los ficheros XML. Si contienen errores de formación, el método build provocará una excepción (JDOMException), que capturándola se puede obtener un mensaje con la razón del error y el fichero y la línea del mismo en la que se ha producido (que se mostrará por pantalla y se concatenará en el atributo de log) Después, se procederá de la misma forma con la instancia de SAXBuilder inicializada con el descriptor del fichero XSD.

Por cada mensaje con errores de formación detectado se incrementará un contador de errores de este tipo, y por cada mensaje con errores de validación se incrementará otro contador.

Finalmente, el método debe determinar si hay mensajes aptos para cargar, devolviendo true si los hay y false si no. Esto se determina comprobando que la diferencia entre el número total de ficheros que contenga el directorio menos el número de ficheros ignorados sea mayor que 0, mayor que el número de mensajes con errores de validación y mayor que el número de mensajes con errores de formación.

Desde el programa principal de este módulo sólo se llamará al método de carga de mensajes si este método de validación ha devuelto true.

6.3.4.5 cargarMensajes

Método principal de la herramienta, es el que más operaciones realiza, que más tiempo de ejecución y memoria consume y el que más dedicación ha requerido para su implementación.

Recibe como argumentos la ruta del directorio con los mensajes a cargar, y el nombre del ensayo clínico al que pertenecen los datos de dichos mensajes. Previamente a su llamada, desde el main se inicia una transacción a partir del atributo session para poder realizar cambios en la base de datos, y cuando la ejecución del método finalice y se produzca un retorno al main, se finalizará la transacción, persistiendo sus cambios.

Su procedimiento general es ir analizando los elementos de cada mensaje (ignora directorios , ficheros sin extensión .xml, y ficheros .xml que no se hayan validado completamente), mediante una instancia de la clase Document de JDOM que los represente. Esta clase está comentada en el punto anterior, se obtiene a partir del método build de la clase SAXBuilder.

Se obtendrá el elemento raíz de cada mensaje (etiqueta ClinicalDocument) y se irá recorriendo el contenido de los mismos extrayendo elementos hijos (a partir de su nombre y el valor del espacio de nombres de los mensajes: `urn:hl7-org:v3`) y creando instancias de las clases del CDM que les correspondan, dando valor a sus atributos con los valores que tomen los atributos de los elementos del mensaje. Una vez se ha rellenado completamente el objeto, se hará uso del método save, del atributo session.

En primer lugar, antes de entrar en el bucle que recorra todos los mensajes se genera un objeto de acto para el ensayo clínico, introduciendo el nombre recibido como parámetro en su identificador.

Después, para cada mensaje se empezará analizando la cabecera, creando un acto de tipo fichero (classCode=”FILE”) con los datos del documento. Se creará una instancia de ActRelationship para todos estos actos que representan al documento relacionándolos con el acto del ensayo clínico. A continuación se generarán los objetos representativos del paciente, analizando el elemento recordTarget. Se generarán instancias de Entity, LivingSubject y Person. Se creará una instancia de Participation de cada paciente con el acto del fichero creado anteriormente.

Posteriormente se procesará el cuerpo de los mensajes, analizando el contenido de todos sus elementos de la etiqueta entry. Se han incluido todas las estructuras posibles dentro de los elementos entry definidas en el XSD, ya que este método se ha implementado fijándose en el contenido del mismo.

Para el resto de actos (observaciones, procedimientos o administraciones de sustancias) o entidades (medicamentos, material genético) que se generen además de los ya comentados se generará un identificador aleatorio, mediante los métodos definidos en la siguiente subsección 6.3.4.6.

Para cada nuevo acto generado se creará una participación con la entidad representativa del paciente del mensaje (generada a partir de los datos de la cabecera), y una relación con el acto representativo del mensaje (acto con classCode=”FILE” generado también a partir de los datos de la cabecera).

A la hora de insertar nuevas entidades de cualquier tipo se comprueba previamente que no estén ya insertadas, haciendo uso de un objeto Criteria a partir de la sesión para realizar consultas SELECT a tabla Entity (explicado anteriormente en las subsecciones correspondientes a otros métodos). Si una misma entidad, por ejemplo un medicamento, figura en varios de los mensajes a cargar no se cargará de nuevo pero sí se generará un nuevo objeto de las tablas Role y Participation, de modo que en la base de datos resultante la entidad de dicho medicamento tendrá asociado un rol distinto dentro de cada participación con cada acto de administración de sustancia en el que intervenga.

Si se va a cargar un mensaje con mismo identificador que uno ya cargado o uno con distinto identificador de mensaje pero relativo a un mismo paciente ya cargado, se saltará la parte de generación de objetos que representen los datos de la cabecera, es decir, se mantendrá el acto de información del documento y la información demográfica del paciente, añadiéndoles nuevos actos relacionados.

6.3.4.6 s4 y generate_Id

Métodos auxiliares para generar cadenas aleatorias como identificadores de actos o entidades. El método s4() genera un número hexadecimal de 4 cifras y el método generate_Id() concatena el resultado de repetidas llamadas al método s4(), concatenando también el delimitador "-" entre ellos.

El método generate_Id() es invocado en numerosas ocasiones desde el método de carga de mensajes, cada vez que crea un acto nuevo. A la hora de crear entidades o roles en su lugar llama al método s4() repetidas veces, para generar un identificador sin delimitadores.

6.3.5 Esquema XSD para la validación de mensajes conforme al estándar HL7 CDA

Este componente se ha desarrollado para la implementación del método para validar mensajes (subsección 6.3.4.5) descrito en el apartado anterior. Por esta razón, inicialmente formaba parte del módulo principal, pero se separó del mismo para facilitar su acceso desde el fichero JAR ejecutable en el que se comprimirá la herramienta (esta cuestión se explicará en detalle en la siguiente sección, 6.4).

El camino seguido para la implementación de la funcionalidad de validar mensajes siguiendo las directrices del estándar HL7 CDA de composición de los mismos ha sido desarrollar un fichero XSD (*XML Schema Definition*) en el que se definan todas las estructuras aceptadas en los mensajes y todos los elementos que pueden aparecer en ellos (así como los hijos que puede tener cada elemento, las ordenaciones entre elementos, sus cardinalidades, etc…), aplicando todas las directrices establecidas por el

estándar, como los tipos de datos para todos los elementos (y de sus atributos), los elementos mínimos obligatorios y los opcionales, etc…

Para la generación del XSD se ha partido de un mensaje HL7 base, generado manualmente. Dicho mensaje se ha generado a partir de los mensajes correspondientes a distintos datasets almacenados en el CDM de los proyectos EURECA e INTEGRATE, procedentes de distintas instituciones colaboradoras de dichos proyectos. En la cabecera del mensaje base se ha insertado información demográfica ficticia a cerca de un paciente, insertando también información ficticia sobre otras estructuras posibles que puede presentar la cabecera como el autor del documento, la propia información del documento, o la organización de la que proviene el mensaje. En lo referente al cuerpo estructurado del mensaje base, con el fin de que cubriera el mayor número de estructuras posibles, se ha insertado en él información diversa extraída tanto de templates (o plantillas) de las wikis de los proyectos EURECA e INTEGRATE[2][3] como de mensajes de distintos datasets utilizados en los proyectos, anonimizando sus valores (dando valores aleatorios a las cantidades, modificando las fechas, etc…). De este modo, el mensaje base contiene una combinación de la gran mayoría de estructuras procesadas hasta la fecha por los servicios utilizados en los proyectos EURECA e INTEGRATE. Se ha tratado de incluir en el mensaje representaciones de las posibles combinaciones de elementos que especifica el estándar, por ejemplo, para que la cardinalidad del elemento targetSite sea mínimo 0 y máximo n (opcional pero que pueda repetirse n veces) y que dentro de él pueda haber un elemento qualifier, que a su vez pueda tener elementos name y value se han insertado tanto actos sin targetSite, como actos con el elemento targetSite repetido, como actos con un elemento targetSite con el elemento qualifier, y así tratando de cubrir todas las posibilidades. Se ha seguido este procedimiento para la mayoría de elementos.

Una vez estaba construido un mensaje base lo suficientemente completo se generó una aproximación inicial del esquema XSD deseado, gracias a la aplicación web "*XSD/XML Schema Generator*" de Freeformatter [37], que dispone de un cuadro de texto en el que insertar el contenido de un fichero XML y genera automáticamente las reglas de definición del esquema que representa a dicho documento.

Pero se han tenido que aplicar multitud de correcciones a dicho XSD original, a continuación se detallan algunas de las más relevantes:

- Modificar las cardinalidades. Los elementos mínimos obligatorios deben tener el atributo "minOccurs" con valor 1, los opcionales con valor 0 y los elementos que puedan repetirse valor 'unbounded' en el atributo "maxOccurs". En el XSD inicial no siempre se tomaban los valores correctos. Por ejemplo, el elemento entry puede estar vacío o bien contener una observación, o una administración de sustancia, o un procedimiento, por tanto se ha establecido como un complexType basado en una propiedad<xs:choice> cuyos elementos tienen minOccurs="0" y maxOccurs="1".

- Para definir elementos que puedan tener una serie de elementos hijos en cualquier orden se ha cambiado su contenido de <xs:sequence> a <xs:all>. Pero esto impedía que cualquiera de sus hijos se repitiera. La solución fue sustituir la propiedad< xs:all> por <xs:choice maxOccurs="unbounded"> [31].

- El tipo asignado a los elementos que representaran códigos de conceptos inicialmente se estableció como entero, ya que la mayoría eran de SNOMED-CT, que asigna códigos numéricos a sus elementos, pero dado que los códigos de LOINC pueden contener delimitadores (“-”) y los de HGNC letras (“ESR1”), se cambió el tipo de los códigos a String.

- Completar los elementos principales, añadiendo todas las estructuras posibles que les faltaban. Como por ejemplo la posibilidad de que dentro del elemento hubiera elementos como targetSite o methodCode.

Aún con todas las correcciones que se han aplicado, el XSD no es perfecto, pero es una aproximación inicial bastante notable a lo que debería ser una implementación en XSD de las reglas del estándar HL7 CDA. Se ha probado a realizar ejecuciones del método de validar mensajes utilizando como entrada todos los mensajes de los datasets utilizados en los proyectos EURECA e INTEGRATE en el último año. Y en base a los resultados se han realizado las correcciones mencionadas anteriormente. Y esta será la forma de proceder en el futuro, ir completando el XSD mediante el *feedback* aportado por nuevos mensajes de estudios que surjan en el futuro, analizando las necesidades de representación que tengan que sean válidas según el estándar y no estén aun contempladas en el XSD. Este es el mejor procedimiento posible para construir el XSD: construir una aproximación inicial del mismo a partir de unos determinados mensajes e ir mejorándolo adaptándolo a nuevos mensajes que se vayan recibiendo. Si no su construcción sería infinita. Por ejemplo, la etiqueta actRelationship debe tener un límite de anidamiento establecido en función de los mensajes que se hayan procesado, (por ejemplo que dentro de la entrada correspondiente a una observación pueda figurar una relación con un procedimiento que esté relacionado con una administración de sustancia, es decir, anidamiento de 3 niveles), ya que si se quisiera implementar un XSD que aceptara cualquier número de anidamientos mediante esta etiqueta, tendría que contener toda la definición de las estructuras de observaciones, procedimientos o administraciones de sustancia infinitas veces dentro de la etiqueta actRelationship de cada una de ellas.

El fichero XSD entregado aún tiene elementos en un estado un poco primitivo, como por ejemplo los elementos autor y custodian de la cabecera, que figuran en él pero no tienen la representación más adecuada en el XSD porque son elementos que aún no se han usado en mensajes procesados en los proyectos EURECA e INTEGRATE, o

elementos con una estructura definida pero que no ha sido contrastada ya que no se dispone de mensajes que la contengan.

6.4 Organización de los ficheros, entrega y guía de ejecución

El proyecto de Java en el que se ha realizado la herramienta contiene el fichero de Reverse Engineering y el fichero de configuración de Hibernate en su paquete por defecto y dos paquetes más, el paquete "CDM" con las clases de las entidades y sus mappings y el paquete "ETL", con la clase principal ETL.java con las funciones descritas anteriormente, la clase FileChooserDemo para la implementación gráfica del explorador de carpetas, un icono necesario para esta clase y el fichero XSD para validar los mensajes según el estándar CDA de HL7 v3.

Además contiene incorporadas las siguientes librerías:

- Conector de MySQL para Java versión 5.1.18
- JDOM 2.0.5
- Hibernate
- Hibernate JPA (capa de abstracción para varias librerías ORM, de persistencia).

La herramienta se entregará en un archivo comprimido denominado "codigo_tfg.rar" compuesto por tres carpetas. Por un lado se entregará una carpeta con el código fuente de la herramienta. En ella se encuentran todos los ficheros que han sido implementados. Dado que la herramienta completa (preparada para ejecutarse) se entregará en un fichero JAR y las clases dentro de él se encuentran compiladas (.class), no se podrían consultar, por tanto se ha tomado la decisión de crear esta carpeta aparte que contiene todas las clases Java previas a su compilación y el resto de ficheros desarrollados (como los ficheros de *mapping*). La utilidad de esta carpeta es únicamente para visualizar el código, pero no es necesaria para el funcionamiento de la herramienta.

La segunda carpeta, denominada "Herramienta", contiene los principales elementos necesarios para ejecutar el programa, que son los siguientes:

- ETL_Project.jar: Fichero ejecutable de la herramienta.

- lib: Carpeta con todas las librerías que necesita la herramienta. Para la correcta ejecución del programa esta carpeta debe colocarse en el mismo directorio que el fichero JAR.

- hibernate.cfg.xml: Fichero de configuración de Hibernate, descrito en profundidad anteriormente. Para el correcto funcionamiento de la herramienta es necesario que el fichero se encuentre en la misma ruta que el JAR. No puede

incluirse dentro del JAR ya que se necesitan hacer lecturas y modificaciones del mismo, y es mucho más sencillo situarlo en la misma ruta que la herramienta y dentro del código de esta crear los descriptores de acceso a este fichero (File, Filewriter) especificando la ruta relativa al directorio actual de trabajo ("./hibernate.cfg.xml"), donde también se encontrará el JAR.

- HL7_CDA.xsd: Fichero XSD de descripción del esquema con todos los elementos posibles de los mensajes HL7 CDA, descrito anteriormente. Se ha extraído del JAR y se han cambiado las rutas de los descriptores de acceso a este fichero por la misma razón que el fichero de configuración de Hibernate, aunque en el caso de este fichero la herramienta sólo necesita hacer lecturas.

- log.txt: Fichero de ejemplo de la salida de una ejecución. Se verá sobrescrito al realizarse una ejecución de la herramienta, ya que ésta genera este fichero en el directorio actual de trabajo al finalizar su ejecución. Para una visualización lo más correcta y legible posible es recomendable abrir este archivo con el editor de texto y de código fuente libre, Notepad++. Se distribuye bajo los términos de la Licencia Pública General de GNU [33].

- leeme.txt: Fichero de instrucciones para realizar una ejecución.

- ModuloPruebas.jar: Implementación de las pruebas del sistema. Requiere el uso de las mismas librerías que la herramienta principal, y realiza lecturas al fichero de configuración de Hibernate, por lo que se ha situado en el mismo directorio. Se debe ejecutar tras realizar una ejecución de la herramienta, y realizará una serie de consultas a la base de datos en la que la herramienta ha insertado los mensajes. Para su ejecución se debe introducir el siguiente comando: `java -jar "ModuloPruebas.jar"` desde el mismo directorio donde se ejecutó la herramienta principal. El módulo de pruebas se explica en mayor profundidad en el capítulo7.

La tercera carpeta, denominada "auxiliar", contiene el resto de ficheros necesarios para realizar una ejecución de prueba, que son los siguientes:

- HL7_MySQL_Schema_2.6.2.sql: Este script permite generar una instancia vacía del modelo 2.6.2 del modelo común de datos para bases de datos relacionales utilizado en los proyectos EURECA e INTEGRATE. La construcción de este script no ha sido realizada por el alumno y no forma parte del trabajo realizado en este TFG, pero se ha incluido ya que es necesario utilizarlo para realizar pruebas con la herramienta.

- PacienteFicticio1.xml y PacienteFicticio2.xml: Dos ejemplos de mensajes HL7 ficticios, utilizados para depurar la herramienta ya que incluyen multitud de

estructuras posibles de los mensajes. Se entregan para posibilitar la realización de ejecuciones de ejemplo de la herramienta.

A continuación se detallan los pasos a seguir para realizar una ejecución de prueba de la herramienta. Los únicos requisitos a nivel de instalación de software son disponer del JRE de Java (versión 1.6 o superior), pues es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java, y disponer, como mínimo de MySQLWorkbench (si sólo se dispone de él, sólo se podrían realizar ejecuciones remotas, creando previamente la conexión en el workbench para dicha máquina remota), y si se quieren realizar ejecuciones locales, MySQL server. Se ha probado con distintas versiones de ambos y en principio no se ha detectado ninguna incompatibilidad con ninguna versión. Si se quiere realizar una ejecución local se debe generar una base de datos mediante la ejecución del script de SQL para el esquema vacío del CDM (proporcionado en la carpeta "Auxiliar"). Se puede nombrar a la base de datos como se desee, con la restricción de que el nombre no contenga caracteres numéricos.

Tal y como se explica en el fichero "leeme", para realizar una ejecución en primer lugar es necesario descomprimir todo el contenido de la carpeta "Herramienta" en una ruta determinada, y desde la terminal situarse en dicha ruta.

Previamente a insertar el comando que inicie la ejecución, es necesario asegurarse de que los componentes con los que la herramienta interactúa estén listos. Si se va a realizar una ejecución local, el servidor de MySQL debe estar arrancado, y disponer de alguna base de datos con la estructura del CDM. Si la ejecución será remota, se debe tener configurada una conexión en MySQL Workbench con la máquina a la que se cargarán los datos extraídos de los mensajes (al generarla, desde Workbench se testea la conexión solicitada, comprobando si la máquina solicitada existe, se encuentra corriendo y se puede establecer una conexión, viendo si es accesible y si los datos de autenticación son válidos). Se debe preparar un directorio que contenga los mensajes HL7 a cargar. Para realizar una ejecución básica de prueba se incluyen dos ejemplos de mensajes HL7 (en la carpeta "Auxiliar"). Estos mensajes se han construido a partir de mensajes reales, pero son totalmente ficticios. Se han extraído trozos de mensajes de distintos estudios para generar un mensaje que tenga mucha variedad de estructuras para poder depurar la herramienta intentando cubrir la mayor parte de sus posibilidades. Pero se ha anonimizado el contenido de los mensajes. Se han cambiado los identificadores por "ficticio1" y "ficticio2", se han falsificado las fechas de nacimiento y las fechas de todos los actos, y se han cambiado los posibles valores que pudieran tener, como valores de observación o dosis de administraciones de sustancias. El hecho de juntar actos que no tienen ninguna relación entre sí, procedentes de mensajes de distintos ensayos clínicos también forma parte de la anonimización, ya que el paciente resultante tendrá un historial clínico resultante absolutamente ficticio, e incluso en algunos casos sin sentido.

Posteriormente se debe introducir el siguiente comando en la terminal: `java -jar "ETL_Project.jar"` y se dará comienzo a la ejecución de la herramienta. A continuación se muestra un listado de los eventos que surgen durante la ejecución en el orden en el que ocurren y se describirá la interacción que realizan con el usuario:

- Se le pregunta al usuario si desea cambiar los parámetros de la conexión. Si introduce "si" (en mayúsculas o minúsculas) se le solicitará que introduzca el valor que desea dar al nombre de la base de datos, usuario y contraseña de MySQL, y el host de la conexión si desea que sea remota. El usuario siempre tiene la posibilidad de mantener el valor por defecto de los campos que no desee cambiar simplemente pulsando la tecla Enter cuando se le solicita que introduzca su valor. Una vez introducidos los parámetros, se crea la sesión estableciendo una conexión a partir de ellos.

- Se le solicita al usuario que determine la ruta que contenga los mensajes que desea cargar, a partir de la interfaz gráfica implementada para el explorador de carpetas. Una vez seleccionado se imprimirá la ruta completa del directorio seleccionado.

- A continuación se analiza el contenido del directorio especificado. Se muestran avisos si el directorio contiene directorios anidados o ficheros que no tengan extensión XML y se realizan dos análisis a los mensajes HL7 contenidos en el directorio. En primer lugar, se buscan errores de formación y en segundo lugar se validan según el estándar de construcción de mensajes HL7 CDA (utilizando el esquema XSD desarrollado). Una vez finalizados los análisis se muestra al usuario el número de mensajes con errores de formación y de validación con respecto al número total de mensajes analizados.

- Si el directorio contiene al menos un mensaje sin errores de validación, se continuará la ejecución, si no, se irá al último punto de esta lista. Se le solicita al usuario que introduzca el nombre del ensayo clínico de procedencia de sus datos, para dar valor al atributo "trial" y poder generar un acto global que represente al ensayo y se relacione con cada acto que represente a un mensaje.

- Se comprobará si la base de datos de destino está vacía, dando las opciones al usuario de vaciar todo su contenido o de mantenerlo.

- Se procederá a la carga de datos, extrayéndolos de los mensajes. Se informará al usuario de qué fichero se está procesando en cada momento.

- Se preguntará al usuario si desea guardar un fichero log con toda la información relevante de la ejecución, incluyendo la que se le ha mostrado por pantalla e información adicional avanzada como los detalles de cada objeto que se

almacene, los avisos y el número de avisos de terminología desconocida surgidos durante la carga de mensajes, el tiempo total de ejecución en minutos, etc…

7 PRUEBAS DEL SISTEMA

En este capítulo se documenta la implementación del módulo de pruebas de la herramienta, que se ha desarrollado para cumplir las fases de validación y depuración de la herramienta, en las que tras una ejecución completa de la herramienta (con todos sus componentes implementados completamente) se comprueban los resultados obtenidos para la detección de errores y posteriores implantaciones de mejoras en la herramienta para optimizarla y reparar defectos.

Este módulo se ha implementado siguiendo uno de los objetivos marcados para este TFG, pues realiza una validación de la herramienta ejecutando una batería de consultas automáticas.

El módulo se ha implementado como un proyecto Java independiente, en el que también se incluyen las clases de representación del CDM implementadas en la herramienta principal, ya que las consultas SELECT que realizará el módulo para comprobar la integridad de los datos de los mensajes también se realizan mediante las facilidades aportadas por Hibernate, es decir, mediante objetos Java y una sesión que genera automáticamente consultas SQL a partir de código Java.

La forma de ejecutar este módulo y validar la herramienta consiste en comparar la salida del módulo de pruebas con los datos contenidos en los mensajes insertados, verificando que se ha almacenado todo su contenido de manera correcta.

Este módulo, entregado comprimido en un fichero JAR ejecutable, se compone del paquete con la implementación en Java del CDM (idéntico al de la herramienta), y una clase con un método main que implementa la sucesión de consultas.

La clase principal contiene dos atributos, factory y session, con la misma función que los presentes en la clase principal de la herramienta, es decir, realizar una lectura al fichero de configuración de Hibernate (se usará el mismo fichero para la herramienta y para el módulo de pruebas, de modo que el módulo de pruebas utilice la misma conexión que la que deja configurada el usuario al ejecutar la herramienta) para poder obtener un objeto de la clase Session de Hibernate a partir de una determinada conexión, que pueda realizar consultas a la base de datos a la que se accede mediante dicha conexión.

Este módulo únicamente realiza lecturas en la base de datos (consultas SELECT), no realiza ninguna modificación, por tanto no requiere la creación de una transacción.

Su implementación se basa en crear objetos Criteria (de Hibernate) para cada una de las tablas de la base de datos mediante el método createCriteria del atributo de la sesión. Estos objetos contienen una lista con todas las entradas registradas en la tabla que les corresponda de la base de datos. Se irán recorriendo cada una de estas listas (mediante un bucle for each) siempre que no estén vacías, imprimiendo un listado con los valores de campos relevantes de cada una de sus entradas, lo cual supone la implementación de las consultas SELECT.

Hay algunos casos particulares, como el análisis del contenido de la tabla Participation, que requiere además el anidamiento de bucles de recorrido de las listas correspondientes a las tablas Entity y Act, ya que en la tabla Participation únicamente figuran los identificadores de dichas tablas, y para mostrar una información más completa se realiza una consulta a múltiples tablas para extraer más información a partir de dichos identificadores (como los títulos de entidades y actos).

El método de vaciar la base de datos implementado en la clase principal de la herramienta ha sido muy útil para la realización de ejecuciones sucesivas durante la etapa de pruebas de la herramienta.

A continuación se muestran capturas de una ejecución del módulo de pruebas tras cargar en la base de datos los mensajes de ejemplo proporcionados en la carpeta "Auxiliar" del código.

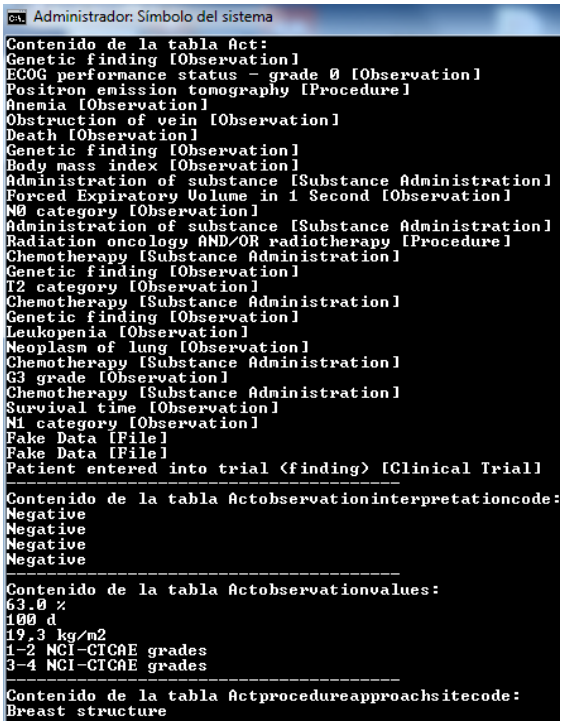
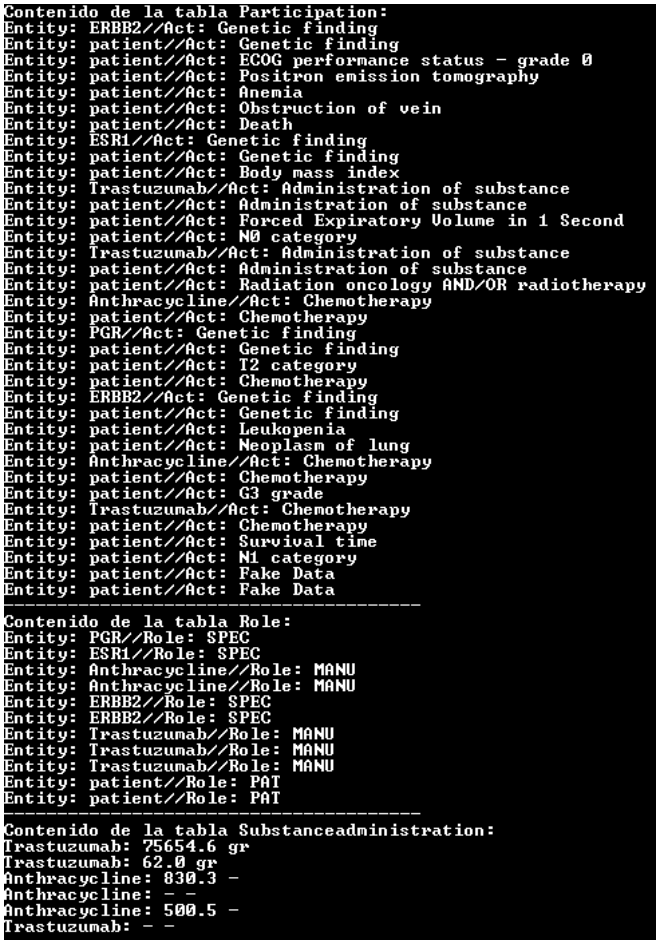


Figura 11. Captura 1 de la ejecución del módulo de pruebas tras cargar los mensajes de ejemplo.



Figuras 12 y 13. Capturas 2 y 3 de la ejecución del módulo de pruebas tras cargar los mensajes de ejemplo.

8 CONCLUSIONES GENERALES

Se ha desarrollado una herramienta que implementa un modelo común de datos basado en el estándar RIM de HL7 v3 empleando técnicas de programación orientada a objetos, y de mapeo objeto-relacional (ORM) de modo que se ha podido realizar toda la interacción necesaria con bases de datos relacionales mediante lenguaje Java.

La herramienta también presenta una fuerte interacción con el lenguaje XML, ya el principal uso que se le da a los objetos con los que se implementa el modelo es la carga de datos extraídos de mensajes XML construidos basándose en el estándar CDA de HL7 para el intercambio de información clínica. Procesando estos mensajes se da valor a los objetos que posteriormente se almacenaran en la base de datos de manera persistente gracias a las funcionalidades aportadas por la herramienta Hibernate, de ORM para Java.

Han sido de especial ayuda las librerías Hibernate y JDOM para Java, claves en el desarrollo de este proyecto. El uso de JDOM ha facilitado muchísimo el procesamiento de ficheros XML de manera muy sencilla.

Gracias a la integración de estos elementos se ha conseguido producir una herramienta que cumple con todos los objetivos marcados para este proyecto. La herramienta, además de la funcionalidad marcada, contiene parte de funcionalidad adicional, que demuestra la gran cantidad de posibilidades que ofrece la programación orientada a objetos en la interacción con bases de datos. Algunos ejemplos son la implementación de una función sencilla que vacía todo el contenido de una base de datos mediante el uso de listas de objetos de todas las entradas de una tabla, que se van recorriendo eliminando todos sus elementos de forma persistente mediante el uso de un objeto de sesión de conexión a la base de datos, o una función semejante que en lugar de eliminar los objetos consulte o actualice los valores de sus campos. En estos ejemplos se acaba generando consultas SQL, que pueden ser algo complejas y se implementan de manera más intuitiva mediante lenguaje Java, ya que, por ejemplo, sólo es necesario implementar un bucle que recorra una lista y se invoque a un método “delete” para cada uno de sus elementos.

Otra gran ventaja de la herramienta es que se ha conseguido que sea dependiente de una única plataforma software, el sistema de gestión de bases de datos relacional MySQL. Con respecto a la idea inicial del proyecto, se ha ganado mucha portabilidad, lo cual aumenta la utilidad y la accesibilidad de la herramienta.

La mayor dificultad que se ha encontrado en el desarrollo de este TFG ha surgido a la hora de estudiar en profundidad el estándar CDA y la guía de implementación del mismo en mensajes, que pueden contener una amplia cantidad de posibles formatos y que deben regirse por un gran conjunto de reglas. Se ha tratado de implementar este estándar en un esquema XSD que permite la validación de mensajes de entrada según el estándar. Dicho XSD consiste en una aproximación incompleta al estándar, pero que trata de cubrir la gran mayor parte de posibilidades de representación que tienen estos mensajes, y que se ha ido mejorando progresivamente (y se continuará mejorando).

9 FUTURAS LÍNEAS DE ACTUACIÓN

Este capítulo tiene como fin mostrar los propósitos establecidos para mejorar la herramienta en el futuro así como los futuros usos que se van a hacer de ella:

- Se ampliará el contenido del fichero XSD de validación de los mensajes y el código del método de carga de datos en función de la recepción de mensajes HL7 generados a partir de nuevos datos de ensayos clínicos, que puedan presentar nuevas necesidades de representación aún no contempladas.

- Se introducirán relaciones en la base de datos del CDM para poder aprovechar al máximo las ventajas de la programación orientada a objetos, especialmente los mecanismos de herencia. Actualmente el CDM presenta relaciones establecidas de forma manual, mediante la asignación del mismo identificador entre dos tablas (por ejemplo en un targetSite asociado a un acto), o mediante el uso de la tabla ActRelationship. En el futuro, introduciendo las debidas relaciones en la base de datos, se podrán establecer las relaciones de una forma más eficiente, reduciendo el número de tablas (por ejemplo pudiendo prescindir de la clase ActRelationship), sustituyéndolas por atributos dentro de las clases, por ejemplo cada acto presentaría un atributo en forma de lista con los identificadores de todos los actos asociados con él, o del mismo modo otro atributo para todos los targetSite asociados a un acto, lo cual facilitaría la implementación del método de carga de datos. Y también se empleará la herencia entre clases (siendo por ejemplo la clase Person hija de la clase LivingSubject, que a su vez sería hija de la clase Entity), pudiendo reutilizar código de clases superiores estableciendo una representación más lógica y más clara del CDM.

- Utilizando las relaciones comentadas en el punto anterior se emplearán para modificar los métodos toString() de las clases Java representativas del CDM de forma que puedan dar una estructura más clara y legible al fichero de log en el que se registra la información de la ejecución. Actualmente se imprimen en él los valores que toman cada uno de los atributos de cada objeto que se va almacenar, pero se intentará mejorar para poder anidar esta información. De forma que, por ejemplo, cuando se muestre la información de un acto se pueda anidar la información de otras tablas, como de sus targetSites asociados. Esto se hace únicamente con los objetos que representan claves primarias compuestas. Ejemplificando, esto es lo que se registra actualmente en el log:

  Act{id:…,code…,…}

  ActTargetSiteCode{id:ActTargetSiteCodeId{id:…,code…,…}, …}

  Y esto es lo que se pretende conseguir:

  Act{…targetSite:ActTargetSiteCode{id:ActTargetSiteCodeId{…}, …}}

  Ó Entity:…{LivingSubject:…{Person:…}}

- Se intentará reordenar el código, reducir el número de creaciones de variables, y el consumo de memoria para tratar de optimizar la herramienta aumentando su

eficiencia y reduciendo su tiempo de ejecución. Tal y como está ahora, ha tardado en almacenar un dataset de 219 mensajes aproximadamente 6 minutos, lo cual es mejorable, a pesar de que dichos mensajes contenían aproximadamente 15 actos cada uno, que es una complejidad ligeramente por encima de la media de los mensajes con los que se ha tratado.

- Se intentará que la herramienta sea compatible con otros sistemas gestores de bases de datos distintos a MySQL. Aunque no suponga una necesidad para los proyectos en los que se enmarca la herramienta (ya que en EURECA o INTEGRATE se emplean únicamente bases de datos de MySQL), no supondrá mucha complicación la inclusión de drivers o librerías y adaptar ciertos aspectos del código para que la herramienta pueda operar con otros sistemas gestores de bases de datos como Oracle o PostgreSQL.

- Se implementará una interfaz gráfica (especificada como *feature* opcional de la herramienta, requisito 6 de la especificación de requisitos), que de la posibilidad de especificar a la herramienta los mensajes a cargar mediante *drag and drop* (arrastrándolos a un cuadro de la interfaz).

- Se integrará la herramienta con el servicio Data push, para crear un cliente web que pueda acceder a la herramienta de manera remota. Y se realizarán ejecuciones remotas de la herramienta, principalmente con el servidor kandel.dia.fi.upm, que almacena todas las bases de datos de estudios clínicos de las instituciones miembros de los proyectos EURECA e INTEGRATE. Pero en estos casos conviene destacar que, a la hora de usar la herramienta en un entorno real, se hará siempre una primera prueba en local, para comprobar si los mensajes a insertar contienen elementos o estructuras válidas según el estándar pero que no estén reconocidas en el código o en el XSD, arreglando estos elementos antes de realizar la ejecución remota.

10 REFERENCIAS

10.1 Información general

[1] Wiki oficial de Health Level Seven International.http://wiki.hl7.org/index.php?title=Main_Page

[2] Wiki del proyecto EURECA de investigación biomédica. Contiene información de las fuentes de los datos y de estructuras tipo de mensajes HL7 CDA, descripción del

CDM y toda la información relevante del proyecto. Disponible en el siguiente enlace (requiere registro): http://atlas.ics.forth.gr/EURECA/wiki/index.php/Main_Page

Plantillas de mensajes HL7 CDA: http://atlas.ics.forth.gr/EURECA/wiki/index.php/A_list_of_available_HL7_v3_templates_to_export_data

[3] Wiki del proyecto INTEGRATE de investigación biomédica. Contiene información de las fuentes de los datos y de estructuras tipo de mensajes HL7 CDA, descripción del CDM y toda la información relevante del proyecto. Disponible en el siguiente enlace (requiere registro): http://atlas.ics.forth.gr/INTEGRATE/wiki/index.php/Main_Page

[4] HL7 International Organization. http://www.hl7.org

[5] *Guía para el desarrollo de documentos CDA (elementos mínimos)*.Subcomité Técnico V3-CDA HL7 Spain. 23/02/2007.

[6] *HL7 Implementation Guide for CDA® Release 2: IHE Health Story Consolidation, DSTU Release 1.1 (US Realm)*. Health Level Seven International. Julio 2012.

[7] *Guía de implementación HL7 del estándar CDA para el almacenamiento de historias clínicas de acuerdo con los requerimientos de la Fundación HL7 Colombia* http://esalud.unicauca.edu.co/wiki/images/1/17/Gu%C3%ADa_de_implementaci%C3%B3n_Odontologia.pdf

[8] *Documentos clínicos electrónicos HL7 CDA R2 - Estructura general*. HL7 en Español. http://hl7es.blogspot.com.es/2011/04/documentos-clinicos-electronicos-hl7.html

[9] *The HL7 Clinical Document Architecture.* National Center for Biotechnology Information, US National Library of Medicine, National Institutes of Health. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC130066/

[10] *Clinical Document Architecture*. Wikipedia http://en.wikipedia.org/wiki/Clinical_Document_Architecture

[11] UMLS (Unified Medical Language System ®) Terminology Services. A service of the U.S. National Library of Medicine, National Institutes of Health. Aplicación web que contiene un navegador de terminologías médicas como SNOMED-CT, Disponible en el siguiente enlace (requiere registro): https://uts.nlm.nih.gov/home.html

[12] *HL7 v3.0 Data Types ITS-XML*. Gunther Schadow, Regenstrief Institute for Health Care http://amisha.pragmaticdata.com/v3dt/ITS-XML.html

[13] Descripción del estándar de prácticas recomendadas para la especificación de requisitos software IEEE Std. 830-1998". FDI-UCM. https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf

[14] Descripción del estándar de prácticas recomendadas para la especificación de requisitos software IEEE Std. 830-1998". Computadores y Tiempo Real, Universidad de Cantabria. http://www.ctr.unican.es/asignaturas/is1/IEEE830_esp.pdf

[15] Ejemplo de especificación de requisitos para un Sistema de Gestión de Paquetes de Préstamos. Ingeniería del Software II Curso 2012/2013

[16] Ejemplo de ERS para un Sistema de Subastas. Ingeniería del Software II, Curso 2013-2014

[17] Especificación de Requisitos Software. Wikipedia. http://es.wikipedia.org/wiki/Especificaci%C3%B3n_de_requisitos_de_software

[18] Ejemplo de diseño software para un Sistema de Administración Docente. Facultad de ciencias empresariales, Universidad de BíoBío, Chile.

[19] Lic. Federico Arambarri. Ejemplo de documento de diseño de arquitectura de software para Nexus Collection, módulo de telecobranzas.

[20] Juan Manuel Moratilla Vargas, investigador de los proyectos EURECA e INTEGRATE, del GIB de la UPM. *Implementación de un modelo relacional basado en el Modelo de Información de Referencia del estándar HL7 v3 para la integración de datos biomédicos, Trabajo de fin de carrera.* Facultad de Informática, UPM. Septiembre 2012.

[21] Corepointhealth.com. HL7 resources: R-MIM-Refined Message Information Models. http://www.corepointhealth.com/resource-center/hl7-resources/r-mim-refined-message-information-model

10.2 Tecnologías empleadas

[22] MySQL (MySQLCommunity Server 5.6.17) http://dev.mysql.com/downloads/mysql/

[23] MySQL 5.7 Reference Manual. http://dev.mysql.com/doc/refman/5.7/en/

[24] Luis Mengual Galán, LSIIS Facultad de Informática, UPM. *Bases de datos: Elementos Básicos de SQL*

[25] NetBeans IDE 7.4 https://netbeans.org/downloads/7.4/

[26] Gavin King with colleagues from Cirrus Technologies. *Tutorial básico de Hibernate*. Traducción al español por David Marco Palao.

[27] Héctor Suárez González (javahispano.org), *Manual Hibernate.*

[28] JDOM Javadocs. Documentación de la librería JDOM de Java para la interacción con XML. http://www.jdom.org/docs/apidocs/

[29] Tutorial de JDOM de java2s.com
http://www.java2s.com/Code/Java/XML/JDOM.htm

[30] Conjunto de tutoriales de JDOM 2.0 por Study Trails.
http://www.studytrails.com/java/xml/jdom2/java-xml-jdom2-introduction.jsp

[31] Solución al problema en las propiedades de los esquemas XSD, de que la propiedad "xs:all" no permite elemento con cardinalidad mayor a 1.
http://stackoverflow.com/questions/3827572/xml-schema-to-match-the-following-all-with-unbounded-maxoccurs/3827606#3827606

[32] Ejemplo de implementación de la clase JFileChooser y sección explicativa de su funcionamiento. Sitio web de documentación de Oracle.
http://docs.oracle.com/javase/tutorial/displayCode.html?code=http://docs.oracle.com/javase/tutorial/uiswing/examples/components/FileChooserDemoProject/src/components/FileChooserDemo.java

http://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html

[33] Notepad plus. Sitio web oficial de descarga libre. http://notepad-plus-plus.org/

[34] Documentación de la clase JFileChooser de Java. Sitio web de documentación de Oracle. http://docs.oracle.com/javase/7/docs/api/javax/swing/JFileChooser.html

[35] Código Java para actualizar un fichero XML usando JDOM. TechBrainwave.com
http://www.techbrainwave.com/?p=391

[36] XSD: XML Schema Definition. Javier Soriano (jsoriano@fi.upm.es). Lenguajes y Sistemas Informáticos e Ingeniería del Software, Facultad de Informática, Universidad Politécnica de Madrid.

[37] XSD/XML Schema Generator. Freeformatter.com
http://www.freeformatter.com/xsd-generator.html

[38] Tipos de datos dentro de un esquema XSD. New Mexico Tech computer center.
http://infohost.nmt.edu/tcc/help/pubs/rnc/xsd.html

[39] XSD Restrictions/Facets. W3Cschools.com

http://www.w3schools.com/schema/schema_facets.asp

[40] XML schema compositors and properties. Oxygenxml
http://www.oxygenxml.com/doc/ug-editor/topics/xml-schema-diagram-compositors-properties.html

[41] Hibernate Factory and Session classes docs. docs.jboss.org

http://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Session.html

http://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/SessionFactory.html

[42] Hibernate tutorial. Red hat middleware, LLC. 2004
http://docs.jboss.org/hibernate/core/3.3/reference/en/html/tutorial.html

[43] Web oficial de Hibernate. http://hibernate.org/

[44] Problema de las palabras reservadas utilizando Hibernate con MySQL.
http://stackoverflow.com/questions/444932/hibernate-generates-invalid-sql-query-with-mysql

[45] *Changing Hibernate schemas in runtime*. forum.hibernate.org
https://forum.hibernate.org/viewtopic.php?f=1&t=948509

[46] Propiedades del fichero de configuración de Hibernate. Web NMS Developer Guide, Persistence Services/Configuring Database Parameters.
http://www.webnms.com/webnms/help/developer_guide/persistence_services/database_parameters.html

[47] *How to use Hibernate in a NetBeans platform application.* DZone SQLZone.
http://netbeans.dzone.com/how-to-nb-hibernate

[48] *Errores comunes en Hibernate*. Hackelare.wordpress.com

http://hackelare.wordpress.com/2010/07/05/errores-comunes-en-hibernate-anexo-1/

[49] Mirth Connect, oficial website. Mirthcorp.com

http://www.mirthcorp.com/products/mirth-connect

[50] Mirth Connect. Wikipedia.

http://en.wikipedia.org/wiki/Mirth_Connect

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Jun 06 22:07:13 CEST 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)